

LNCS 2188

Frank Bomarius
Seija Komi-Sirviö (Eds.)

Product Line Software Improvement

Third International Conference
Kaiserslautern, Germany, September 2000
Proceedings

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Tokyo

Frank Bomarius Seija Komi-Sirviö (Eds.)

Product Focused Software Process Improvement

Third International Conference, PROFES 2001
Kaiserslautern, Germany, September 10-13, 2001
Proceedings



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Frank Bomarius
Fraunhofer IESE
Sauerwiesen 6, 67661 Kaiserslautern, Germany
E-mail: bomarius@iese.fhg.de

Seija Komi-Sirviö
VTT Electronics
P.O. Box 1100, 90571 Oulu, Finland
E-mail: seija.komi-sirvio@vtt.fi

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Product focused software process improvement : third international conference ; proceedings / PROFES 2001, Kaiserslautern, Germany, September 10 - 13, 2001. Frank Bomarius ; Seija Komi-Sirviö (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2188)
ISBN 3-540-42571-3

CR Subject Classification (1998): D.2, K.6, K.4.2, J.1

ISSN 0302-9743

ISBN 3-540-42571-3 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author
Printed on acid-free paper SPIN: 10840575 06/3142 5 4 3 2 1 0

Preface

The Third International Conference on Product Focused Software Process Improvement (PROFES 2001) continued the success of the PROFES'99 and PROFES 2000 conferences. PROFES 2001 was organized in Kaiserslautern, Germany, September 10 - 13, 2001. The PROFES conference has its roots in the PROFES Esprit project (<http://www.ele.vtt.fi/profes/>), but it quickly evolved into a full-fledged general-purpose conference in 1999 and since then it has gained wide-spread international popularity.

As in previous years, the main theme of PROFES 2001 was professional software process improvement (SPI) motivated by product and service quality needs. SPI is facilitated by software process assessment, software measurement, process modeling, and technology transfer and has become a practical tool for quality software engineering and management. The conference addresses both the solutions found in practice as well as relevant research results from academia. The purpose of the conference is to bring to light the most recent findings and results in the area and to stimulate discussion between the researchers, experienced professionals, and technology providers for SPI.

With the tremendous growth of the Internet, e-commerce, m-commerce, telematics, and telecommunication applications, it is ever more important to emphasize the quality of software products and processes. With plenty of new people and new software-based applications emerging at a very fast pace, it is easy to forget the importance of product and process improvement, and to repeat the same mistakes that have been committed in more traditional software development. The PROFES conferences have always addressed this issue by explicitly focusing on mechanisms for ensuring high product quality under various circumstances of software development and for diverse market needs.

As in 2000, another important element of the conference was the Learning Software Organizations (LSO 2001) workshop, which was again organized in conjunction with the PROFES conference. The LSO workshop series is a communication forum that addresses the questions of organizational learning from a software point of view and builds upon existing work on Knowledge Management and Organizational Learning. LSO complements the PROFES program, which encourages fruitful discussions and information exchange between the participants of PROFES 2001 and LSO 2001. The proceedings of LSO 2001 appear as LNCS 2176.

The conference program included four top level keynote speakers (Prof. Dieter Rombach of the University of Kaiserslautern and Fraunhofer IESE, Prof. Victor Basili of the University of Maryland and Fraunhofer-Center Maryland, Prof. Werner Mellis of the University of Cologne, and Prof. Mary Shaw of Carnegie Mellon University – in order of appearance). Once again we received plenty of high-quality submissions. Each paper was reviewed by three independent reviewers. The program committee was very critical in its reviewing and selected 27 papers for presentation at the

conference. In addition, the committee selected three half-day tutorials and one full-day tutorial.

We wish to thank Fraunhofer IESE, the University of Kaiserslautern, VTT Electronics, and the University of Oulu for supporting the conference. We are also grateful to the authors for providing high-quality papers, the Program Committee for reviewing and participating in the design of the program, the Organizing Committee, and numerous individual contributors who helped to organize this conference.

July 2001

Frank Bomarius
Seija Komi-Sirviö

Conference Organization

General Chair

Pasi Kuvaja, University of Oulu, Oulu (Finland)

Organizing Chair

Petra Steffens, Fraunhofer Institut Experimentelles Software Engineering,
Kaiserslautern (Germany)

Program Co-chairs

Frank Bomarius, Fraunhofer Institut Experimentelles Software Engineering,
Kaiserslautern (Germany)
Seija Komi-Serviö, VTT Electronics, Oulu (Finland)

Panel, Workshop, and Tutorial Chair

Ilkka Tervonen, University of Oulu, Oulu (Finland)

Publicity Chair

Petra Steffens, Fraunhofer Institut Experimentelles Software Engineering,
Kaiserslautern (Germany)

Program Committee

Andreas Birk, Fraunhofer IESE (Germany)
Lionel Briand, Carleton University (Canada)
Richard Castanet, Université Bordeaux (France)
Reidar Conradi, NTNU (Norway)
Bärbel Hörger, DaimlerChrysler (Germany)
Hajimu Iida, Nara Institute of Science and Technology (Japan)
Janne Järvinen, Solid Information Technology (Finland)
Ross Jeffery, University of New South Wales (Australia)
Erik Johansson, Q-Labs (Sweden)
Kari Käsälä, Nokia Research Center (Finland)
Karlheinz Kautz, Copenhagen Business School (Denmark)
Graham King, Southampton Institute (UK)
Paolo Nesi, University of Florence (Italy)
Risto Nevalainen, STTF (Finland)
Markku Oivo, Solid Information Technology (Finland)
Tua Rahikkala, Cybelius Software (USA)
Günther Ruhe, Fraunhofer IESE (Germany)
Kurt Schneider, DaimlerChrysler (Germany)
Veikko Seppänen, University of Oulu (Finland)
Forrest Shull, Fraunhofer Center Maryland (USA)
Rini van Solingen, CMG (The Netherlands)
Guiseppe Visaggio, University of Bari (Italy)
Yingxu Wang, University of Calgary (Canada)
Isabella Wieczorek, Fraunhofer IESE (Germany)
Claes Wohlin, Blekinge Institute of Technology (Sweden)
Matias Vierimaa, VTT Electronics (Finland)
Otto Vinter, Bruel & Kjaer (Denmark)

Table of Contents

Keynote Addresses

Guaranteed Software Quality.....	1
<i>Dieter Rombach</i>	
Building an Experience Base for Software Engineering: A Report on the First CeBASE eWorkshop.....	3
<i>Victor Basili</i>	
A Contingency Approach to Software Development.....	4
<i>Werner Mellis</i>	
Career-Long Education for Software Professionals: A US View of the Educational Challenges in a Rapidly-Changing Technology.....	5
<i>Mary Shaw</i>	

Quality of Software

Cognitive Structures of Software Evaluation: A Means-End Chain Analysis of Quality.....	6
<i>Bernard Wong, Ross Jeffery</i>	
Requirements Evolution from Process to Product Oriented Management.....	27
<i>Stuart Anderson, Massimo Felici</i>	

Software Process Assessment and Improvement

A Case Study on Scenario-Based Process Flexibility Assessment for Risk Reduction.....	42
<i>Josef Nedstam, Martin Höst, Björn Regnell, Jennie Nilsson</i>	
Contexts in KM Based SPI: The Case of Software Design Experience Reuse.....	57
<i>Veikko Seppänen, Seija Komi-Sirviö, A. Mäntyniemi, Tua Rahikkala, Minna Pikkarainen</i>	
Models and Success Factors of Process Change.....	68
<i>Marion Lepasaar, Timo Varkoi, Hannu Jaakkola</i>	
Process Improvement in Turbulent Times: Is CMM Still an Answer?.....	78
<i>Karl Lebsanft</i>	

Assessment of Maintenance Maturity in IT Departments of Public Entities: Two Case Studies.....	86
<i>Macario Polo, Mario Piattini, Francisco Ruiz, Mar Jiménez</i>	

Evaluating a Usability Capability Assessment.....	98
<i>Netta Iivari, Timo Jokela</i>	

Organizational Learning and Experience Factory

Building an Experience Base for Software Engineering: A Report on the First CeBASE eWorkshop.....	110
<i>Victor Basili, Roseanne Tesoriero, Patricia Costa, Mikael Lindvall, Ioana Rus, Forrest Shull, Marvin Zelkowitz</i>	

Experience Magnets: Attracting Experiences, Not Just Storing Them.....	126
<i>Kurt Schneider</i>	

Improving Knowledge Management in Software Reuse Process.....	141
<i>Timo Kucza, Minna Nättinen, Päivi Parviainen</i>	

Processes and Knowledge Management: A Symbiosis.....	153
<i>Christof Nagel</i>	

Augmenting Experience Reports with Lightweight Postmortem Reviews.....	167
<i>Torgeir Dingsøy, Nils Brede Moe, Øystein Nytrø</i>	

A Tool for Managing Software Development Knowledge.....	182
<i>Scott Henninger, Jason Schlabach</i>	

Industrial Experiences and Case Studies

Starting Improvement of Requirements Engineering Processes: An Experience Report.....	196
<i>Marjo Kauppinen, Sari Kujala</i>	

A Family-Oriented Software Development Process for Engine Controllers.....	210
<i>Karen Allenby, Simon Burton, Darren Buttle, John McDermid, John Murdoch, Alan Stephenson, Mike Bardill, Stuart Hutchesson</i>	

Enabling Local SPI in a Multi-national Company.....	227
<i>Peter Fröhlich, Horst Lictor, Manfred Zeller</i>	

Project Improvement as Start-Up.....	240
<i>Ton Dekkers</i>	

Software and Process Modeling

LIPE: A Lightweight Process for e-Business Startup Companies Based on Extreme Programming.....	255
<i>Jörg Zettel, Frank Maurer, Jürgen Münch, Les Wong</i>	

Evaluation of the E3 Process Modelling Language and Tool for the Purpose of Model Creation.....	271
<i>M. Letizia Jaccheri, Tor Stålhane</i>	

Describing Fractal Processes with UML.....	282
<i>Harald Störrle</i>	

Extending the Software Process Culture - An Approach Based on Groupware and Workflow.....	297
<i>Renata Araujo, Marcos Borges</i>	

Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling.....	312
<i>Ulrike Becker-Kornstaedt</i>	

Modular Process Patterns Supporting an Evolutionary Software Development Process.....	326
<i>Michael Gnatz, Frank Marschall, Gerhard Popp, Andreas Rausch, Wolfgang Schwerin</i>	

Empirical Software Engineering

A Classification Scheme for Studies on Fault-Prone Components.....	341
<i>Per Runeson, Magnus C. Ohlsson, Claes Wohlin</i>	

Program Understanding Behavior During Estimation of Enhancement Effort on Small Java Programs.....	356
<i>Lars Bratthall, Erik Arisholm, Magne Jørgensen</i>	

Evaluation of a Business Application Framework Using Complexity and Functionality Metrics.....	371
<i>Hikaru Fujiwara, Shinji Kusumoto, Katsuro Inoue, Toshifusa Ootsubo, Katsuhiko Yuura</i>	

Author Index	381
---------------------------	-----

Keynote Address:

Guaranteed Software Quality

Prof. Dieter Rombach

University of Kaiserslautern & Fraunhofer IESE
Kaiserslautern, Germany

Quality requirements for software embedded in technical products such as airplanes or automobiles, or supporting critical business applications such as banking transactions or telecommunications increase steadily. Customers do not only expect high quality, they also expect more and more quality guarantees. In order to guarantee quality of any engineering artefact one needs to understand the desired quality in measurable terms, apply engineering processes which produce such quality, and perform quality control measures regarding process adherence and product quality.

Software engineering lacks well-understood and manageable methods and processes. Well-understood means that the effects of methods and processes under varying context characteristics such as different life-cycle models, experiences of personnel or tool support on product qualities of interest are known. This implies understanding the cause (i.e., process) effect (i.e., product) relationship. Managed means that for a given project context processes can be chosen so that they guarantee expected product qualities. The need for such tailored engineering processes has been eloquently stated by Watts Humphrey in a recent keynote at the SEPG-2000 in India titled 'What if my life depended on it?' He showed empirical evidence that software produced by ad-hoc (i.e., not well-understood) processes can never be guaranteed reliable based on any amount of testing.

Software development is a human-based design process. As such it has a large number of characteristics which vary from project environment to project environment. This represents the specific challenge in understanding the cause effect relationship for software. Knowledge regarding the effects of methods or processes under typical characteristics of a project environment can therefore only be accumulated based on data from that very environment. This kind of environment-specific knowledge based on process-product feedback is called 'experience'.

In this presentation, I will present the specific knowledge (better experience) management challenges for establishing process-product relationships for software engineering. In particular, I will survey a portfolio of methods (e.g., QIP) and organizational schemes (e.g., Experience Factory) for building such experience-based process-product models, give examples of industrial applications (e.g., via Fraunhofer IESE),

and introduce example industry consortia for sharing the load of building up such models (e.g., German ViSEK project, SEC project)

Overall, the messages will be that Software quality can only be guaranteed via well-understood engineering methods and processes. Understanding requires ‘experience-based’ model building – implying that each industrial organization needs to invest! Such approaches have lead to sustainable improvements of predictability (and guarantees) of software quality in industry.

Engineering is defined as ‘controlled use of technology’. If we want to take software engineering seriously we need to be able to understand, manage and control key software methods and processes as suggested in this presentation.

Keynote Address:

Building an Experience Base for Software Engineering: A Report on the First CeBASE eWorkshop

Prof. Victor Basili

Fraunhofer Center for Experimental Software Engineering Maryland

New information is obtained by research and disseminated by papers in conferences and journals. The synthesis of knowledge depends upon social discourse among the experts in a given domain to discuss the impact of this new information. Meetings among such experts are, however, expensive and time consuming. In this paper we discuss the organization of CeBASE, a center whose goal is the collection and dissemination of empirically-based software engineering knowledge, and the concept of the online workshop or *eWorkshop* as a way to use the Internet to minimize the needs of face-to-face meetings. We discuss the design of our eWorkshop and give the results of one eWorkshop that discussed the impact of defect reduction strategies.

(For the full paper see page 110.)

Keynote Address:

A Contingency Approach to Software Development

Prof. Werner Mellis

University of Cologne, Germany

For several years now there is a growing quest for speed and flexibility in software development. It is associated with recommendations for the design of software development like extreme programming, rapid development, agile, adaptive development, synch-and-stabilize, light methodologies etc. Like older recommendations these more recent ones are not explicitly restricted to certain conditions of application. Therefore we can assume that they are considered universally valid by their proponents. As there are several good reasons for expecting the various partly contradictory recommendations not to be equally suitable for any condition, we propose to set up a contingency approach, associating different situations to appropriate recommendations. Several questions arise: Which attributes are used to describe the alternative designs of software development? Which relevant alternative designs can be distinguished? Which attributes are used to describe the alternative situations? Which relevant alternative situations can be distinguished? Which design of software development is recommended for which situation? The empirical investigation of the factual design of software development in different firms demonstrate, that also in a given situation, there is a considerable range of acceptable designs. Therefore a contingency approach also needs to describe and explain the range of design acceptable for a given situation?

Keynote Address:

Career-Long Education for Software Professionals: A US View of the Educational Challenges in a Rapidly-Changing Technology

Prof. Mary Shaw

Carnegie Mellon University, USA

Software's increasingly critical role and the rapid rate of change of associated technologies present serious challenges for maintaining the professional skills of software developers. Current professionals must rely on existing opportunities for mid-career education. Longer-term solutions include improvements in existing opportunities and changes to university education that enhance students' ability to learn new material. A variety of certifications will provide evidence of a professional's current proficiency. I will report on the way these issues are being addressed in the United States.

Cognitive Structures of Software Evaluation: A Means-End Chain Analysis of Quality

Bernard Wong¹ and Ross Jeffery²

¹ Department of Information Systems, University of Technology, Sydney,
PO Box 123, Broadway, NSW 2007, Australia
bernard@it.uts.edu.au

² School of Computer Science & Engineering, University of New South Wales,
Sydney, NSW 2052, Australia
rossj@cse.unsw.edu.au

Abstract. This paper reports on a study of eight interviews conducted in an Australian Internet/Telecommunications organization. We sought to study the stakeholders' understanding of software quality, and the level of importance perceived in regard to different characteristics of software quality. The research finds that different stakeholders have different views of software quality. The research also finds that desired values and consequences sought by the stakeholder influence their view of quality and their choice of product characteristics used in their quality evaluation.

1 Introduction

Adopting an appropriate quality assurance philosophy has often been viewed as the means of improving both productivity and software quality ([1], [2]). However, unless quality is defined, it is very difficult for an organization to know clearly whether it has achieved quality.

Perceptions of software quality are generally formed on the basis of an array of cues. Most notably, these cues include product characteristics ([3], [4], [5], [6], [7], [8]). The cues are often categorized as either extrinsic or intrinsic to the perceived quality. Simply, intrinsic cues refer to product characteristics that cannot be changed or manipulated without also changing the physical characteristics of the product itself, extrinsic cues are characteristics that are not part of the product [9]. Price and brand are thus considered to be extrinsic with respect to product quality.

This paper explores the desired values, the consequences and the characteristics related to software quality evaluation. It examines the outcomes accruing to the users' and developers' perception of software quality. Assuming that these outcomes are instrumental in achieving an individual's desired values [10], knowledge of how quality impacts on the users' or developers' personal values can increase our understanding of software quality measurement. The work is based on a qualitative study of four

users and four developers who all have high involvement with the organization's billing system. Participants of the study were asked to evaluate the quality of the billing system. Following their evaluation, in-depth interviews were used to collect data on what characteristics of the system contributed to their evaluation, and details of the reasons why those characteristics were used. The results of the study are presented in two cognitive structures, one for the users and one for the developers, and illustrate how the two camps differ in both their measure of quality, and what influences the measures.

2 The Meaning of "Quality"

The term "quality" is applied to virtually all products, businesses, professions and processes. It has many definitions, each with a different perspective. Robert Pirsig [11] comments "that quality is easy to see and is immediately apparent when encountered, but when you try to pin it down or define what it is, you find the concept is elusive and slips away". He states that "it is not the complexity, but the simplicity of quality that defies explanation" [11]. He sees quality as an abstract description. Garvin[12] describes this view of Pirsig as a transcendental view, that is, "You know it when you see it, but you can't describe what it is - it is a gut feeling". He describes this view as the one often taken by philosophers, and that if you were an engineer, or an economist, your view would be different. Garvin[12] describes other views of quality - the product-based view, the manufacturing-based view, the economics-based view and the user-based view. He shows how each of these views can be used to define product quality. Engineers who believe a product has set characteristics often adopt the product-based view. These characteristics are used as the measure of quality. The manufacturing-based view is adopted when one believes the quality development process determines a quality product. Recently, many organizations have obtained ISO9000 certification [13]. Certification assesses the manufacturing process and is awarded on successfully having a quality management system in place. The economics-based view is adopted by economist who believe that price has a correlation with quality. And lastly, the user-based view is the one which emphasizes that each individual will have their own perception of quality. Garvin[12] states that most existing definitions of quality fall into one of these categories be it conformance to specification, or meeting user requirements or best practice.

Pfleeger [14] supports this, stating that software quality is determined by the person analyzing the software and that people have different views of quality. The reasons behind why ones view and perception differ can be found in the studies performed in applied psychology. For example, the psychologist Kelly, quoted in Weiner[15], asserted that "meaning" is subject to change and depends on the eye of the beholder. Kelly believed that each person is an individual with his or her own views. A person's perception is therefore guided by how he or she understands and interprets the world at large. Since individuals perceive the same situation and experiences in different ways, it follows that their perceptions of quality will also differ. Users, for example

may judge software to be of high quality if it does what they want in a way that is easy to learn and easy to use. Software developers may judge software on how well the program is written, the choice of algorithms used, the efficiency and speed of the program. Software can be judged by those who are designing and writing code, by those who maintain the programs after they are written, by those who use the software and by the managers who pay for the software to be developed. Therefore, software quality, contain factors and characteristics which address the needs of users, developers, maintainers and managers, and these may differ from one person to the next ([14], [16], [17]).

Researchers in the software engineering area have tried different ways to define quality. They have adopted the product-based view ([3], [4], [5], [6], [7], [8]), the manufacturing-based view ([13], [18], [19], [20], [21], [22]), the user-based view ([23], [24], [25], [26], [27], [28], [29], [30], [31]) and even combinations of the views ([32]).

The models which follow the product-based view, define quality by a collection of characteristics and factors ([3], [4], [5], [6], [7], [8]). Though these models have the same view of quality and have similar characteristics, they differ in the number of factors and criterias. For example, Boehm's model has seven characteristics [3], McCall's has eleven characteristics [5], Bowen's has thirteen characteristics [8], Murine's has twelve characteristics [6], and the recent international standard for Information Technology software product evaluation, ISO9126 [33], has six characteristics ([34], [35]). Kitchenham [8] made a number of observations about these product-based models. She stated that there seemed to be little supporting rationale for including or excluding any particular quality factor or criterion. Definitions of quality factors and criteria were not always consistent when comparing one model to the next. It appears that though the product-based view of quality has been widely used and accepted, the differences between them, highlights how ones view can dictate the perception of quality. These models, when used by people with different backgrounds and assumptions, can result in various interpretations for a particular characteristic.

The manufacturing-based approach has been another focus in the software engineering area. Its inception had been largely encouraged by the introduction of software process improvement practices ([13], [18], [19], [20], [21], [22]). The approach follows the manufacturing industry where the basic belief is that the quality of the finished product is determined and measured by the process used to create it ([20], [21], [22]). It is expected that by improving the development process, improvements in product quality will reduce re-work during development, and reduce maintenance effort after the products have been delivered, consequently reducing overall life cycle costs.

This manufacturing-based approach was initially promoted during the early 1980s by a small group of industrial and academic software engineers ([21]). Though this approach has much merit and has been applied successfully in a number of cases ([20], [21], [22]), it is not guaranteed that by trying to introduce new practices to improve the software development process, that one will obtain a quality product ([35]).

Constantine [35] states that there is a danger of only focusing on the development processes and the technical practices, and losing sight of the people who must master and carry out the development processes. Even with such systematic approaches as the Software Engineering Institute's Capability Maturity Model, unless new practices are fitting to the people who will put them into practice, real improvement may remain elusive [35].

Recent research has adopted the user-based view of quality ([23], [24], [25], [26], [27], [28], [29], [30], [31], [32]). The view recognizes that each person is different and takes on a different perception of quality.

The SQUID model, an ESPRIT 3 SQUID project was the result of a reassessment by Kitchenham of what constituted a quality model [32]. Kitchenham built SQUID based on expert opinion about internal software properties that are likely to influence external properties, that it was important to monitor and control internal measures, which may influence external properties. Based on the McCall and ISO 9126 models, SQUID aimed to use measurable properties to link internal properties and external characteristics to quality characteristics. Kitchenham states that the underlying principle for SQUID is that software quality requirements cannot be considered independently from a particular software product. Kitchenham acknowledges her alignment with Gilb [36] on this point when he states that quality requirements arise from the operational properties required of the software, or of its support/maintenance environment. The SQUID model comes far closer than any in being able to express the balance of need for internal and external view, and considerations of the change in characteristics and metrics that are involved with each type of software in each context.

Vidgen ([29], [31]) proposed a framework to define quality based on the Multi-view development method ([37], [38], [39]). Vidgen believed that multiple perspectives of software quality are required if one is to assess product quality properly. The framework was based on customer satisfaction, relating the product with its use and the services provided to support it. These three views provide the basis for the multiple perspectives. It was not merely an exercise in looking at the same object from different angles but entailed different assumptions about what quality was.

Though the topic of software quality has been around for decades, software product quality research is still relatively immature, and today it is still difficult for a user to compare software quality across products. Researchers are still not clear as to what is a good measure of software quality because of the variety of their interpretations of the meaning of quality, of the meanings of terms to describe its aspects, of criteria for including or excluding aspects in a model of software, and of the degree to which software development procedures should be included in the definition ([40]).

There have been many studies on the topic of software quality, yet little empirical studies on what influences the perception of quality for different people. Our earlier research ([16], [17]) concluded that different groups of people view quality in differ-

ent ways, and that it was possible to group people with similar definitions of quality and similar choices of characteristics in their quality assessment process. However, the studies did not address what influenced the choice of characteristics used in quality evaluation. This paper focuses on whether the desired values sought by the evaluator determines their choice of characteristics for the quality evaluation, and whether people with the same desired values can be grouped together as having similar definitions of quality and similar sets of characteristics.

Personal values are beliefs people have about important aspects of themselves and the goals toward which they are striving. Personal values are the penultimate consequences of behavior for people, be their feelings of self-esteem, belonging, security, or other value orientations. As such, personal values are part of the central core of a person. That is, personal values determine which consequences are desired and which are undesirable.

Values have been shown to be a powerful force in influencing the behaviours of individuals in all aspects of their lives ([41], [42]). It is proposed in this research, that their use in software quality evaluation shows the behavior of software evaluators and the relationship between the software characteristics, the desired consequences, and the values sought. Several attempts in consumer research have been made to provide a theoretical and conceptual structure connecting consumers' values to their behavior ([10], [43], [44], [45]). The basis by which the study is performed, is via adopting Gutman's means-end chain model ([10], [46], [47], [48], [49]), which posits that linkages between product characteristics, consequences produced through usage, and personal values of users underlie the decision-making process, or in our case the software quality evaluation process. The term "means" refers to the software product and services, and "ends" refers to the personal values important to the user.

Gutman's model is able to give a complete representation of the means-end chain, representing linkages from the characteristics to the values, along with the capability of explicating the chain. Earlier models ([43], [44], [45]) either focused on only the consequences and values, without the characteristics or attributes which are related to them, or the model does not encourage explicating the means-end chain. The means-end chain model seeks to explain how a person's choice of a product or service enables him or her to achieve his or her desired result. Such a framework consists of elements that represent the major usage processes that link personal values to behavior. Two assumptions underlie this model:

- all user actions have consequences; and
- all users learn to associate particular consequences from the usage of the product or service.

Consequences may be desirable or undesirable; they may stem directly from usage or the act of usage, or occur indirectly at a later point in time or from others' reaction to ones' consumption behaviour. The central aspect of the model is that users choose actions that produce desired consequences and minimize undesired consequences. Of course, because it is the characteristics that produce consequences, consideration for

the characteristics that the software product possesses must be made. Therefore, it is important to make aware the characteristic-consequence relations. Overall, the characteristic-consequence-value interrelations are the focus of the model. Values provide the overall direction, consequences determine the selection of specific behaviour in specific situations, and the characteristics are what is in the actual software product that produce the consequences. It is knowledge of this structure that permits us to examine the underlying meaning of quality.

The goal of this paper is to study the structural relations among the characteristics used to determine software quality in relation to the desired values and consequences. The research question being addressed is whether the desired values sought by the quality evaluator determines the choice of characteristics used in the evaluation.

3 Study Context

Eight subjects were interviewed singly and in-depth at an ISP telecommunications organization during early 2000. All respondents were involved with the software being evaluated, either as a user of the software or as a developer supporting the software. Four of the subjects were users and four were developers. The organization is an international telecommunications company. It is among the world's premier voice and data communications companies, serving consumers, businesses, and government. With annual revenues of more than \$62 billion and 160,000 employees, the organization provides services to customers worldwide. In Australia, only the internet products are available. They offer dialup services, web hosting, and a variety of proprietary data processing services. With the growing competition in the internet area, many different combinations of services and products have been introduced. Discounting through volume sales and bundling of products have been very popular. As such, the current billing system, which automates the complex calculations required to create the invoices, is needed to cater for all products, services, and packages, in an accurate, flexible and efficient way. The success or failure of the business can be as a result of this billing system.

The people surveyed came from different jobs and backgrounds in this organization. All the respondents had reasonable experience with the billing system. Most of them have been in the current job for at least 2 years. We surveyed two programmers, a systems analyst, and a project manager from the development side. The development team all had a minimum of 4 years experience in the I.T industry. We surveyed the financial controller, who "owns" the system, the billing clerk, and two debt collectors, who all use the system.

The interviews aimed to focus on the respondent's perception of software quality and why they assess the software with the characteristics they did. There were no hints, nor guidelines used during the interview, which would influence the subjects to give any particular result.

4 Data Collection and Analysis

To identify the full set of linkages connecting means to ends, users were given a laddering task ([50], [51], [52]). The laddering procedure consists of a series of directed questions based on mentioned distinctions the individual has with respect to the quality of the software being evaluated. The purpose of the laddering is to force the user or developer up the “ladder of abstraction” to uncover the structural aspects of user knowledge as modeled by the means-end chain. The questioning procedure was designed around the unique demands of the laddering procedure. They were based on prior answers, the interpretation of the answers, and focused on “pushing the participant up” the characteristic-consequence-value hierarchy. Very often, further clarification of the answers was sought before introducing another question. The laddering method has been widely used in the consumer research and psychology disciplines. It has become an accepted method to gain insights regarding sources of value that are perceived to be, or could become, motivationally important ([10], [46], [47], [48], [49]).

As a warm-up, subjects were asked about their interest in computers, in particular software. This was followed with questions about their frequency of involvement with the billing system and whether or not they like the software. Respondents were asked to explain how they could tell if the software was of high quality or not. Next, subjects were asked to rate the quality of the billing system, that is, whether the system was high quality or not. The subjects were asked to describe why the quality rating and what were the influences which led them to that score. The interview would then focus on each reply, with continual questions based on each answer given. The questions and answers created a chain starting from the characteristic used in the quality evaluation, with links to the desired consequences, until finally ending with the value sought. This process would be repeated continuously for each identified characteristic, resulting in a number of “ladders” being created for each respondent.

The following description illustrates the laddering process and gives a clearer description of the procedure used for eliciting the elements. Melissa, the billing clerk, was asked about the aspects that came to mind when evaluating the billing system. Probing proceeded by asking her to explain how she could tell if the billing system was of high quality or not. The following excerpt is part of the interview transcript with her. The underlined phrases highlight significant content, with insertions of identified characteristics, consequences and values shown in bold italics.

Melissa: Well, this billing system works and does not fall over!! I like it.

Characteristic identified ...”System works and does not fall over”

Interviewer: Why? Is this important?

Melissa: The previous billing system used to be a dog ... it would do as it pleased. Sometimes it would work fine, but other times ... I would spend hours waiting for an invoice run, with the system appearing to run smoothly, but to discover the data to be wrong, or worse still, it would run till it was nearly finished, and then it would just hang or fall over with a non comprehensible message. I would have to redo everything. And sometimes it would fall over again.

Interviewer: How did this affect you?

Melissa: It was a waste of my time. I had no time for anything else. The boss thought I was lazy and unfair comments would be made. It seemed I always had an excuse for not doing anything else other than the billing run. The software made me look bad!! I had to stay back after work, and sometimes work over the weekends. It was horrible. I was so stressed, I couldn't sleep, I was moody, and my weekends would be ruined. I was always in a rush because time was always against me.

Consequences identified ... Time for other things; Save Time; Not feel rushed; Looks Good at Work

Interviewer: That's terrible ... so this system affects your job security.

Melissa: I guess it does. This new system makes me look good because I can do my job quickly, whereas the old system ... well!!

Value identified ... Job Security

Interviewer: I gather from what you've said that you hate staying back after work, or working over the weekend.

Melissa: Don't get me wrong. It's okay if it's once in a while, but not when it was as frequent as it was. The weekends are my time. I don't want to be at work. I'm there five days, and that's long enough. I value my personal time away from work, especially time with my family. It's my time. I don't want to waste my life spending it all ways at work!! What sort of life is that?

Consequences identified ... Spend Time with family

Interviewer: So, are you saying that the quality of life is important and it is affected by this system?

Melissa: Well, come to think of it, yes.

Value identified ... Better Life (Quality of Life)

Interviewer: And the stress ... how does this affect you?

Melissa: Well it affects my health, which ruins my quality time with the family. I'm always tired because I don't sleep properly. I'm tense and irritable. I tell you, after the billing run has worked, properly, I am so relieved and relaxed.

Value identified ... Better Life (Quality of Life)

After performing the interviews, the transcripts were analyzed. The first step in the analysis was to conduct a thorough content analysis of all the elicited concepts. All the responses at the characteristics level were considered first, so that terms close in meaning could be grouped together. The goal here was to reduce the fragmentation of responses that occurred when respondents were using their own language or terminologies, without losing meaning, by grouping elements with widely divergent meanings into the same category. This procedure was repeated at the consequence and value levels. All laddering responses were then expressed in a set of standard concepts. The aggregate set represented the content component of the respondents' quality evaluation structure. The results are then represented in tables, and then modeled using a structured chart, one chart for the users, and one for the developers.

5 Results

Any instance in which a subject links at least two elements together in an asymmetrical fashion (A causes, produces, or leads to B) is defined as a ladder. In all, 31 ladders were elicited from the eight subjects, the shortest ladder having a length of two and the longest ladder a length of seven. The typical ladder was comprised of from three to five elements, although there were many two-element ladders (that is, when a number of characteristics lead to the same consequence).

Three tables for users and three tables for developers are represented here. It would be trivial to address the issue of just the differences between users and developers. The difference is obvious. However it is of interest to identify what the desired values and consequences are for each, and to determine whether they are the influences for their choice of characteristics.

The first tables, table 1 and table 4, list the characteristics collected from the respondents during the interviews. Content analysis was used to simplify this list, reducing any redundancies. The characteristics were grouped using headings found in other quality models and found in software engineering literature. These headings also support the internal and external cues, as defined by Olson and Jacoby [9]. The second tables, table 2 and table 5, list the consequences identified during the laddering

process and the third tables, table 3 and table 6, list the desired values sought by the respondents, which were linked to the consequences in the laddering process.

Table 1. Content Analysis of Characteristics FROM THE USERS

GROUPING of the CHARACTERISTICS	CHARACTERISTICS
Support Driven Perspective	Level of support from I.T. Level of support from developers Online Help Documentation to refer to
Economically Driven Perspective	Cost of software Cost of development Cost of support
Institutionally driven Perspective	Brand name Reputation of Consulting company References from past jobs
Functionally driven perspective	Does what I need it to do Results are correct and accurate Required reports are obtainable quickly Queries about clients are obtainable
Ease of Use driven perspective	Easy to Use Messages on screen are clear and easy to follow Looks nice and attractive Lots of help on screen Good, clear report Tasks are performed with a single button – no effort on my part
Operationally driven perspective	System is available when needed Software never falls over with error System is reliable – always there when I need it. System is efficient, quick response, without delays Results of reports and queries are always correct

Table 2. Content Analysis of Consequences Obtained FROM THE USERS

CONSEQUENCES
Flexible/can do my job easily in different ways
Can do job faster/quicker/saves time
Not feel rushed at work/can take my time
More options to choose from/not limited in how the system works
Convenient
Feel comfortable
Can do more/can do better
Time for other things
Can do more/can do better
Easier to make decisions/make up mind
Look good

Table 3. Content Analysis of the desired Values Obtained FROM THE USERS

VALUES
Better Life – Quality of life
Security – Job is secure

Tables 1, 2 and 3 show the list of characteristics, consequences and values obtained from the set of users. As can be seen, the number of elements obtained is lengthy. It was discovered during the interviews that the desired consequences and values focused on the individual’s job and how the job affected his or her personal life.

The users all focused on valuing “Better Life” and “Job Security”. No matter which characteristic was being discussed, the desired consequences and values would always lead to these two values.

The characteristics elicited related to support, economics, functionality, usability and operationally driven issues. No characteristics relating to technical issues, like development process, programming language used, system documentation, were raised by the users. The respondents all seemed to start with the operational characteristics “System is available when needed” or “Software never falls over with error”. This highlighted the users’ bad past experience when systems failed. Stories of lost weekends, sacrificed due to system failures and incorrect results from the past systems were emotionally raised. Stories of managers showing dissatisfaction on their job performance were also highlighted with the hint that their careers were being affected, even though it was not their fault but the fault of a poor system.

The consequence of “TIME” was also continuously being raised. “Flexible/can do my job easily in different ways “, “Can do job faster/quicker/saves time“, “Not feel rushed

at work/can take my time“, “Time for other things “, all focused on wanting the system to save time. Respondents highlighted that time would take away “Quality of Life” as much as ruin job security, that they look bad if they took too long to create a report. If the system was not flexible, time would be wasted. If the system failed to give correct reports or results, time would be wasted with corrections, reruns, and seeking help from IT. All this would increase stress, and health, resulting in poor quality of life.

Characteristics about functionality, usability and support were identified quite quickly after the operational characteristics. The respondents unanimously expressed their views on the importance of functionality and how it affected their job, usage of time, stress, and therefore quality of life, and job security. It was interesting that at this stage the users focused on different aspects of the system. They were all driven by what their jobs involved. For example, the billing clerk kept highlighting the billing run and how long it would take, the accuracy of the rating or calculations required on the invoices, whilst the debt collector focused on the wonderful speed for querying past transactions, and the ease by which he is able to track payment history. The users raised the problems they had with usability, or ease of use. Comments about having functionality, but not having usability were raised as being a major problem in some past systems. The chief accountant, for example, raised the issue of staff taking ages to enter data into the system, that redundancies existed, and that much time was wasted in mistakes caused by poorly designed screens. The characteristic “support” was also raised by the chief accountant as a major issue. He believed that if the system is difficult to use, then support becomes very important. If it is easy to use, there is very little need for support.

Other characteristics were raised, such as cost or value for money, brand name, and reputation. Not much discussion was given to these characteristics, though the users all stated that these characteristics led to having a system, which delivered better quality of life and better job security.

Table 4. Content Analysis of Characteristics FROM THE DEVELOPERS

GROUPING of the CHARACTERISTICS	CHARACTERISTICS
Support Driven Perspective	Level of support from software supplier, e.g. Oracle, Sybase Good technical documentation Good technical list of known product bugs
Technically driven perspective	Well documented code Use of modeling tools like UML, Use Case etc Appropriate choice of programming language Good design with reusability Easy to correct and maintain Database is normalized appropriately Complete ER diagram is available Design allows additional products to be added easily
Functionally driven perspective	Easy to maintain/enhance Works on any computer – NT, UNIX, PC & MACS Reports and queries are what the user wants
Ease of Use driven perspective	Users never get confused with messages on the screen Users are never confused about what to do Error logs and error messages are easy to follow for maintenance
Operationally driven perspective	Never falls over SQL Queries are efficiently written SQL queries are fast Incomplete parameters does not cause the system to fall over, but to give an appropriate message Batch jobs create error logs, audit trails so that DBA, Tech staff know when the job has failed

Table 5. Content Analysis of Consequences Obtained FROM THE DEVELOPERS

CONSEQUENCES
System will last longer
Flexible/can do my job easily in different ways
Fun to go to work/challenging
Not feel rushed at work/can take my time
Need to update system less frequently
Convenient
Easy to maintain/enhance system
Have to maintain/enhance system less
Feel comfortable
Time for other things
Can do more/can do better
Job is not boring/stays interesting
Less conflict/safer/more secure
Impress others/others think I can do my job well/be admired
Easier to fit in with peers
Feel good about self
In-control

Table 6. Content Analysis of desired Values Obtained FROM THE DEVELOPERS

VALUES
Self Esteem
Self Confidence
Accomplishment/Success
Enjoy Life

Tables Figure 4, 5 and 6 show the list of characteristics, consequences and values obtained from the set of developers interviewed. Like the list in figure 1, the number of elements obtained is lengthy. Like the users, the developers focused the desired consequences and values on their jobs and how the job affected their personal life.

However, the desired values differed from the users. At first it appeared as if “Better Life”, a user value, was also being sought, then as the developers were questioned further, it became apparent that developers valued “Enjoy Life”. This was a surprise as it was assumed that all users and developers would seek “Quality of Life” as the value sought. The developers highlighted their fulfilment found in having programming challenges. They sought satisfaction for being able to solve difficult or impossible programming tasks. The developers all highlighted the joy they felt when users gave approval and acknowledgement for their development. In questioning the developers further it was discovered that “Accomplishment/success”, “Self-esteem”, and “Self-confidence” were also driving forces behind their evaluation of software quality.

Success in the delivery of new software applications, boosted the developers' self-confidence and self-esteem. No matter which characteristic was being discussed, the desired consequences and values would always lead to these values.

Unlike the users, technical characteristics played a very important role in the evaluation of software. This supported the findings of earlier studies ([16], [17]). Much of the developers' focus, were on development and programming design and approaches, the development process, program documentation and tools. The discussions centred round the problems faced when lack of adequate processes, documentations and tools. Frustration was raised when discussion moved towards maintenance and enhancements of poor quality software. Similar comments from a number of the developers highlighted the lack of enjoyment in their job when appropriate programming practices were not followed.

Like the users, the developers found functionality, usability and support to be important. However the focus of the developers were different from the users. Rather than support for the developed application, the developers would be looking at support from the manufacturers, for example, the hardware manufacturers, the database supplier. Rather than looking at functionality sought by the users' job, developers were more concerned with functionality for maintenance and enhancements, easy portability between different pieces of hardware, which they needed to support. Rather than ease of use referring to the users being able to do their job easier, the developers were more concerned with having happier users, so that there would be fewer complaints from the users. Of course, having software, which does not fail, has been identified as important. In all, it is quite obvious from the interviews that what is closest to the hearts of developers are technical issues, even when one considers characteristics such as operations, support, functionality and usability.

The developers gave no consideration to characteristics such as cost, value for money, brand name, or reputation. Whilst these characteristics, also had very little interest from the users, perhaps they would have more interest from management.

As a result of the findings listed in the above six tables, the following cognitive structures are presented. Because this study focuses on whether the values and consequences are the determining factor for whether a characteristic is used in quality evaluation, figures 1 and 2 concentrate only on the links aggregated between the consequences and values, and do not illustrate the details within the characteristic layer.

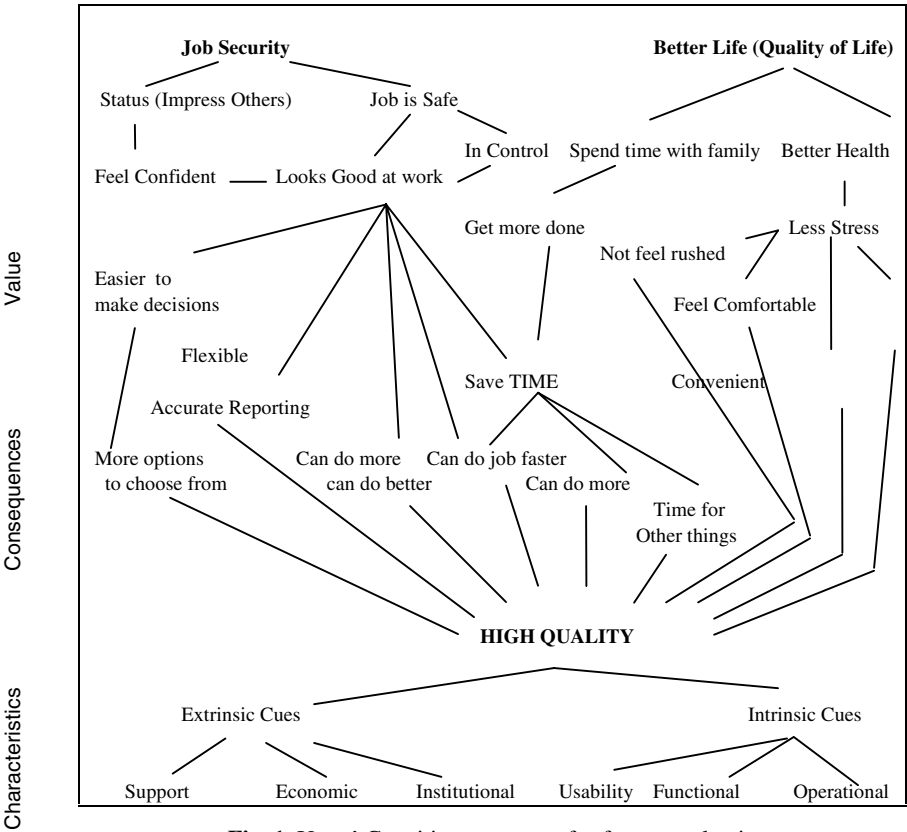


Fig. 1. Users' Cognitive structure of software evaluation

ent meaning or interpretations. Basically, both structures have three fundamental blocks of relations connected to the value level, though each structure differs in what makes up the three blocks.

The principles, underlying the research offers potential application. Given that higher levels of abstraction, namely consequences and personal values, contribute substantially to the bases of preference when selecting characteristics, the means-end framework offers a way of better understanding how inherent product benefits connect with satisfying personal values. This presents an opportunity for gaining a more complete view of what characteristics is perceived as appropriate measure for software quality.

6 Conclusion

The goal of this research was to develop an understanding of what underlies the choice of characteristics used in software quality evaluation. To accomplish this goal, a framework for identifying the various representations was needed. This was accomplished by identifying the bases for selecting characteristics in each of the two groups. The framework imposed on the results, was that of distinguishing individual responses in terms of three broad classes of elements: characteristics, consequences and value as listed in tables 1, 2, 3, 4, 5 and 6. The results elicited the various combinations of characteristics, consequences and values for each person. The results indicate the presence of these types of cognitive elements when dealing with software quality evaluation behaviour. Finally, by aggregating the individual points of view, two cognitive structures, figures 1 and 2, were constructed, which reveal the common preference pathways connecting characteristics to values.

Software engineering literature, have identified many characteristics, in the many models of quality. Though some are similar, many of the models differ in the characteristics, which define quality, with no explanation of what determines the choice of characteristic. It may be, that the definition for quality is different from person to person, as shown in earlier studies of Wong & Jeffery ([16], [17]). The research finds that the desired values sought by the quality evaluator, determines the choice of characteristics used in the evaluation. The users seem to be driven primarily by the need for a better quality of life and better job security. The developers, on the other hand, seem to be more internally driven, either in terms of providing reinforcement of self-esteem and self-confidence, or via a need for enjoying life through accomplishments and success. The research also finds further support for requiring a multi-view approach to evaluating software quality. The two cognitive structures show quite clearly how users and developer differ. They differ, not only in their choice of characteristics, but also in their desired consequences and desired values. It would appear that there are a number of definitions for quality, with all definitions valid.

The use of these cognitive structures can better the understanding of the characteristics selected during quality evaluation, and how they are related to what the evaluator

values. And finally, the research shows the application of Gutman's Means-End Chain Analysis being applied to software quality.

The study has only investigated users of an application and the developers of that application. It has not looked at whether developers, who normally utilize the application as users, would be any different, or whether other job areas, like management would be any different. Further studies are required to address these areas.


References

1. Hatton, L.: The Automation of Software Process and Product Quality, *Software Quality Management I*, Computational Mechanics Publications, 1993.
2. Myers, W.: Debating the many ways to achieve Quality, *IEEE Software*, March 1993.
3. Boehm, B., Brown, J. & Lipow, M: Quantitative Evaluation of Software Quality, Proceedings of the Second International Conference on Software Engineering, 1976, pp592-605.
4. Cavano, J. & McCall, J.: A Framework for the Measurement of Software Quality, *Proceedings of the ACM SQA Workshop*, Nov 1978, pp 133-139.
5. McCall, J., Richards, P. & Walters, G.: Factors in Software Quality, Vol 1,2, & 3, November 1977.
6. Carpenter, S., Murine, G.: Measuring Software Product Quality, *Quality Progress* May 1984, pp16 - 20.
7. Sunazuka, T., Azuma, M. & Yamagishi, N.: Software Quality Assessment Technology, *IEEE 8th International Conference on S.E.*, 1985
8. Kitchenham, B. & Walker, J.: The Meaning of Quality, *Software Engineering 86: Proceedings of BCS-IEE Software Engineering 86 Conference*, Southampton England September 1986.
9. Olson, J.C. & Jacoby, J.: Cue Utilization in the Quality Perception Process", *Proceedings*, 3, M. Venkatesan, ed., Iowa City, Iowa: Association for Consumer Research, 167-79.
10. Gutman, J.: A Means-End Chain Model Based on Consumer Categorization Processes, *Journal of Marketing*, 46 (Spring): 60-72.
11. Pirsig: *Zen and the art of Motorcycle Maintenance: an enquiry into values*, 1974.
12. Garvin, D.: What does "product quality" really mean?, *Sloan Management Review*, 1984, 24.
13. Smilie, T.: Standards for Australia, *Informatics*, Aug 1993.
14. Pfleeger, S.: *Software Engineering: Theory and Practice*, Prentice Hall, 2001.
15. Weiner: *Human Motivation: metaphors, theories, and research*, Sage Publications Inc, 1992.
16. Wong, B. & Jeffery, R.: Quality Metrics: ISO9126 and Stakeholder Perceptions, *Proceedings of the Second Australian Conference on Software Metrics*, 1995, 54-65.
17. Wong, B. & Jeffery, R.: A Pilot Study of Stakeholder Perceptions of Quality, *Technical Report*, CSIRO, 1996.
18. Humphrey, W.: Characterizing the Software Process: A Maturity Framework, *IEEE Software*, 5(2), March 1988, pp73-79.
19. Paulk, M., Curtis, W., Chrissis, M.: *Capability Maturity Model for Software*, Report CMU/SEI-91-TR-24. SEI, Carnegie Mellon University, 1991.

20. Ould, M.: Software Quality Improvement through Process Assessment - A view from the UK, *IEEE Colloquium on Software Quality*, 1992.
21. Dowson, M.: Software Process Themes and Issues, *2nd International Conference on the Software Process: Continuous Software Process Improvement*, 1993, pp54-60.
22. Coallier, F.: How ISO9001 fits into the Software World, *IEEE Software*, Jan 1994.
23. Kitchenham, B.: Towards a constructive quality model Part 1: Software quality modeling, measurement and prediction, *Software Engineering Journal*, July 1987.
24. Kitchenham, B. & Pickard, L.: Towards a constructive quality model Part 2: Statistical techniques for modeling software quality in the ESPRIT REQUEST project, *Software Engineering Journal*, July 1987.
25. Delen: The Specification, Engineering, and Measurement of Information Systems Quality, *Journal of Systems and Software*, 1992, **17(3)**, pp205-217.
26. Eriksson: Quality Function Deployment: a Tool to Improve Software Quality, *Information and Software Technology*, 1993, **35(9)**, pp491-498.
27. Thompson, R.: Quality Agreements for Quality Systems, *Proceedings of Australian Computer Society, Victorian Branch, Annual Conference*, 1993.
28. Juliff, P.: Software Quality Function Deployment, *Software Quality Management II Vol 1*, Computational Mechanics Publications, 1994.
29. Vidgen, R., Wood, J. & Wood-Harper, A.: Customer satisfaction: the need for multiple perspectives of information system quality, *Software Quality Management II Vol 1*, Computational Mechanics Publications, 1994.
30. Dromey, G.: "A Model for Software Product Quality", *IEEE Transactions on Software Engineering*, Vol 21, No. 2 Feb. 1995, pp.146-162.
31. Vidgen: A multiple perspective approach to information systems quality, *University of Salford*, 1996.
32. Kitchenham, B., Lonkman, S., Pasquini, A. & Nanni, V.: "The SQUID approach to defining a quality model", *Software Quality Journal* **6**, (1997) pp.211-233
33. ISO9126: Information Technology Software Product Evaluation - Quality Characteristics and Guidelines for their Use, *International Organisation for Standardisation*, Geneva, 1991.
34. Sanders, J., Curran, E.: *Software Quality*, Addison-Wesley, 1994.
35. Azuma, M.: Towards the 21st Century's Software, State of the art and International Standard in JTC1/SC7, *Software Quality & Productivity*, Dec 1994.
36. Constantine, L.: Fitting Practices to the People: Corporate Culture and Software Development, *American Programmer*, December 1994.
37. Gilb, T.: "Level 6: Why We Can't get There from Here", *IEEE Software*, Jan. 1996, 97-103. *COMPCON 1995 Proceedings*, pp. 52-60.
38. Avison, D. & Wood-Harper, A.T.: Multiview: An Exploration in Information Systems Development, Blackwell Scientific Publications, Oxford, 1990.
39. Wood-Harper, A.T. & Avison, D.: Reflections from the Experience of Using Multiview: through the lens of Soft Systems Methodology, *Systemist*, 14(3): 136-145.
40. Wood, J.R.: Linking Soft Systems Methodology (SSM) and Information Systems (IS), *Systemist - Information Systems Special Edition*, 14(3): 133-135.
41. Comerford, R.: "Software", *IEEE Spectrum*, Jan. 1993, pp. 30-33.
42. Rokeach: Beliefs, Attitudes and Values, San Francisco: Jossey Bass.
43. Yankelovich: New Rules, New York: Random House.
44. Howard: Consumer Behaviour: Application of Theory, New York: McGraw Hill Book Company.

45. Vinson, Scott and Lamont: "The Role of Personal Values in Marketing and Consumer Behaviour", *Journal of Marketing*, 41 (April), 44-50.
46. Young and Feigin: "Using the Benefit Chain for Improved Strategy Formulation", *Journal of Marketing*, 39 (July), 72-74.
47. Gutman, J.: Means-End Chains as Goal Hierarchies, *Psychology & Marketing*, **14** (6):1997 545-560.
48. Bagozzi, R.: Goal-directed behaviours in marketing: cognitive and emotional, *Psychology & Marketing*, **14**, Sept 1997, pp 539-543.
49. Valette-Florence, P.: A Causal Analysis of Means-End Hierarchies in a Cross-cultural Context: Methodological Refinements, *Journal of Business Research*, v 42 No 2 June 1998, pp161-166.
50. Bagozzi, R. and Dabholkar, P.: Discursive psychology: an alternative conceptual foundation to means-end chain theory, *Psychology & Marketing*, **17**, July 2000, pp 535-586.
51. Gutman, J. & Reynolds, T.J. An Investigation of the Levels of Cognitive Abstraction Utilized by Consumers in Product Differentiation, in *Attitude Research Under the Sun*, J Eighmey, ed, Chicago: American Marketing Association, 128-50.
52. Reynolds et al: Laddering Theory, Method, Analysis, and Interpretation, *Journal of Advertising Research*, Feb/Mar 1988, pp.11-31.
53. Reynolds, T.J. and Gutman, J.: Developing Images for Services through Means-End Chain Analysis, in *Emerging Perspectives on Service Marketing*, 1983, 40-44.

Requirements Evolution from Process to Product Oriented Management

Stuart Anderson and Massimo Felici

LFCS, Division of Informatics, The University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ, United Kingdom
{soa, mas}@dcs.ed.ac.uk

Abstract. Requirements Evolution represents one of the major problems in developing computer-based systems. Current practice in Requirement Engineering relies on process-oriented methodologies, which lack of product features. The resulting scenario then is a collection of general methodologies, which do not take into account product features that may enhance our ability in monitoring and controlling Requirements Evolution. This paper shows empirical investigations of two industrial case studies. The results point out evolutionary product features and identify an Empirical Framework to analysing Requirements Evolution. This work represents a shift from process to product-oriented management of Requirements Evolution.

1 Introduction

The requirements process is one of the most important of the entire system life cycle. Most of the faults into computer-based systems (the term computer-based systems draws attention to the involvement of human participants in most complex systems) can be traced backup to the early requirements stage when a recovery action can still be cost-effective [4, 5]. Hence the management of requirements is an important activity for Software Process Improvement (SPI). It is furthermore a key action of many maturity models applied in industry (e.g., CMM [16]). Despite this *Requirements Evolution* [1, 2, 10, 11, 18, 23] still remains one of the most critical issues for software organizations and is challenging for research and practice in Requirements Engineering [14, 25].

Current practice in Requirements Engineering [20, 22, 27] relies on general processes (e.g., elicitation, verification, etc.) and methodologies (e.g., standards and templates for specifying requirements) that are adapted according to specific needs by plugging in other general processes (e.g., task analysis) and methodologies (e.g., formal methods). This view can be really articulated and complex depending on the industrial context. Thus within industrial contexts Requirements Engineering practice is a collection of processes and methodologies driven by stakeholders. The resulting

* This work has been partially funded by a grant of the Italian National Research Council (CNR) within the thematic Science and Information Technology, Bando n. 203.15.11. Research Program: “Requirements Evolution: Understanding Formally Engineering Processes within Industrial Contexts”.

view lacks in taking into account product features that may enhance our ability in engineering requirements for evolving systems. Current practice in engineering requirements demands a shift from process to product-oriented engineering [26]. Unfortunately this shift in Requirements Engineering has not yet been fully registered. The lack of product perspectives in managing Requirements Evolution gives rise to issues degrading both software process and product in terms of dependability. This view becomes even more complex within safety-critical contexts in which evolution can lead to undependable situations [15, 24] causing loss.

Our work is based on investigations of live industrial contexts to devise product-oriented approaches supporting Requirements Evolution. This paper shows a product-oriented analysis of Requirements Evolution in two industrial case studies. The paper is structured as follows. Section 2 introduces the two case studies. The research methodology in Section 3 has driven the empirical analyses presented in Section 4. Section 5 summarises the conclusions and identifies further work.

2 Industrial Case Studies

This section introduces two industrial case studies, an avionics and a smart card case study, which have contributed to our work. The investigation of different industrial case studies is motivated by our research hypotheses. Our work aims to characterise industrial settings in order to identify product-line methodologies [26] for Requirements Evolution. Origins of Requirements Evolution can be identified in stakeholder [18] interaction, which characterises the industrial context, and technical issues [10, 14, 23]. The link among stakeholder interaction, technical issues and Requirements Evolution is still vaguely understood. The analysis of different case studies aims to identify product features that may enhance our ability in monitoring and controlling Requirements Evolution to devise computer-based systems. Our interest is in differences and commonalities across industrial contexts. This aims to define taxonomy of evolution in industrial contexts, which may be characterized by evolving patterns for product-lines. This work contributes to identify dependable Requirements Evolution in terms of process and product, that is, dependable process for deploying evolving systems and Requirements Evolution enhancing system dependability. The following subsections introduce the two industrial case studies, which provide interesting aspects due to their stringent requirements (e.g., safety and security) and to their product-line nature.

2.1 An Avionics Safety-Critical Case Study

The description of the avionics case study identifies critical features with respect to evolution. We describe the case study in general terms, because our focus is on methodologies. On the other hand the case study provides practical issues that may interest practitioners and researchers. The following features are identified in the case study.

Safety Requirements. A system safety assessment analyses the system architecture to determine and categorise the failure conditions. Safety related requirements are determined and flowed down to the software and hardware requirements.

Functional and Operational Requirements. The customer provides the system requirements, which contain information needed to describe the functional and operational requirements for the software. This includes timing and memory constraints and accuracy requirements where applicable. Requirements also contain details of inputs and outputs the software has to handle with special reference to those where non-standard data formats are used.

Software Development Process. The bulk of the software development task can be split into two broad areas, that of software design and code and the other of verification. Two main elements serve to complicate the situation. Firstly there are feedback loops occurring due to problems or modifications. Secondly there is an expansion of information down the design chain to the code. As design progresses, the software requirements are partitioned into smaller more manageable items. This fragmentation is also reflected into further down activities (e.g., testing) and deliverables (e.g., code). There is a strict policy for configuration management, which also requires maintaining traceability on the project to ensure that the final code is complete and consistent with its higher-level requirements, and the verification has been performed on the correct standard of the final code. Software verification consists of two main elements testing and review/analysis. At the start of the project, the verification activities have been planned in specific documents detailing the level of verification performed at each phase of the software development.

Product-Line Aspects and Standards. Hardware dependent software requirements arise not from the system requirements directly but from the implementation of those system requirements. This is normally due to the way that the hardware has been designed to meet its requirements and as an indirect result of safety related requirements. The hardware dependent software requirements characterise the specific product-line in terms of hardware constraints and safety requirements. A certification plan for software is the primary means used by the designed authorities to assess whether the software development process proposed is commensurate with the software level proposed. The plan of the case study has been produced according to the specific guidelines in the standard RTCA/DO-178B [21].

Evolution and Maintenance. During the development life cycle, changes and modifications arise causing feedback loops. As the size and the scope of these changes will be different for each modification and for each project, only general guidelines have been defined. For modifications to a certified standard of software the full life cycle process of testing and reviews should be complied with. The extent to which all activities have to be repeated will depend upon the time elapsed since certification. Modifications that are introduced prior certification will be incorporated during the development life cycle.

2.2 A Smart Card Case Study

People use smart card systems in their daily life. Credit cards, Pay-TV systems and GSM cards are some examples of smart card systems. Smart card systems provide interesting case studies of distributed interacting computer-based systems. Behind a simple smart card there is a complex distributed socio-technical infrastructure. Smart card systems are

Real-Time Systems. The transactions of smart card systems occur in real-time with most of the services operating on a 24-hour basis. The availability of smart card systems is fundamental to support business (e.g., e-money) and obtain customer satisfaction.

Interactive Systems. Most of smart card systems operate on demand. The request of any service provided depends on almost random human factors. Operational profiles show that human-computer interaction turns to be one of the critical factors for smart card systems.

Security Systems. Smart card systems often manage confidential information (e.g., bank account, personal information, phone credit, etc.) that need to be protected from malicious attacks.

The considered smart card context is certified according to many quality and security standards. Among these its management process conforms to PRINCE2. PRINCE¹ (PRojects IN Controlled Environments) is a structured method for project management [6]. It is used extensively by the UK Government and is widely recognized and used in the private sector, both in the UK and internationally. PRINCE2 is a process-based project management approach integrating a product-based project planning. The investigation of the smart card context aims to identify general aspects in requirements management. We do intend neither to validate any particular methodology nor to assess the specific industrial context. Our aim is to identify general practical aspects that can improve our ability in dealing with Requirements Evolution.

3 Empirical Research Methodology

Our research methodology consists of two main steps named *Empirical Analysis* and *Product-oriented Refinement*. The Empirical Analysis aims to gather information from the industrial case studies by analyses of industrial data (e.g., requirements document, data repository). The Product-oriented Refinement is to focus our empirical methodology in order to gather further information by product-oriented analyses.

The analyses consist of both qualitative and quantitative approaches. The qualitative approaches consist of stakeholder interviews focusing on requirements engineering practice (e.g., requirements management policy) in the industrial context

¹ PRINCE is a registered trademark of CCTA (Central Computer and Telecommunications Agency).

and product features (e.g., requirements evolution). Stakeholder interviews are also integrated with a Requirements Engineering Questionnaire [3] consisting of 152 questions organized in terms of *Business*, *Process* and *Product* viewpoints. Qualitative analyses are furthermore performed on documents (e.g., requirements documents, history of changes, repository, etc.) by inspections [9]. Finally, quantitative approaches (e.g., Software Metrics [8]) aim to identify additional information and to support empirical results obtained by qualitative analyses.

We perform incremental empirical analyses and product-oriented refinements, that is, the empirical analyses are incrementally conducted by product-oriented refinements. The better our understanding of the case study, the better our ability in defining product-oriented analyses. The empirical results point out which product-oriented refinements to be implemented in subsequent analyses.

4 Empirical Analyses

This section shows the results of the empirical analyses of the case studies. The analyses investigate different aspects of the case studies depending on the industrial context and the available data. The analysis of the avionics case study aims to identify product features characterising Requirements Evolution, whereas the analysis of the smart card case study aims to assess viewpoints for Requirements Evolution management.

4.1 Avionics Case Study

The empirical investigation of the avionics case study is based on analyses of data repositories of requirements evolution. The aim is to identify requirements properties [1, 2] that may enhance our ability in controlling Requirements Evolution [1]. The investigation goes from a general viewpoint towards a product-oriented viewpoint. The incremental investigation is summarised in what follows.

General Requirements Evolution. Figure 1 shows the total number of requirements changes, i.e., added, deleted and modified requirements, over the 22 software releases². The trend of requirements changes does not give enough information about evolutionary features, but it emphasises the problem of Requirements Evolution. The analysis of Requirements Evolution points out that requirements changes are not uniformly spread out over the three basic changes (i.e., added, deleted and modified requirements), in fact the total number of requirements constantly increases over the software releases. This is because the requirements become clearer to the stakeholders, who split complex requirements into smaller and more precisely stated requirements. Another reason is that new requirements arise during the progress of the project, because there are requirements that cannot be defined at the beginning due to

² There is a correspondence one-to-one between versions of the requirements specification and software releases.

lack of information. Finally, design, implementation and testing activities provide additional feedback to the requirements.

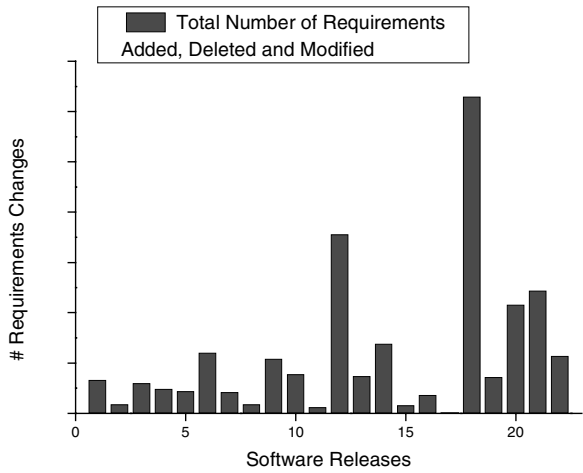


Fig. 1. Number of requirements changes per software release

Taxonomy of Requirements Evolution. The inspection of the history of changes points out the specific changes occurring into requirements. Table 1 shows the taxonomy of requirements changes in the case study. Changes affect requirements attributes like variables, functionality, explanation, traceability and dependency. These attributes are easily collected and represented by requirements templates (e.g., [19, 20]). These attributes are usually embedded within paragraphs specifying requirements. A structured way of representing, collecting and organising requirements attributes may be useful to identify information for controlling and monitoring Requirements Evolution.

Table 1. Taxonomy of requirements evolution

Type of Changes		
General	Domain Specific	Product Line
<ul style="list-style-type: none">– Add, delete and modify requirements– Explanation– Rewording– Traceability	<ul style="list-style-type: none">– Non-compliance– Partial compliance	<ul style="list-style-type: none">– Hardware modification– Variables range modification– Add, delete, rename parameters/variables

Changes fall into three main classes named *General*, *Domain Specific* and *Product Line*. This rough classification may help to identify requirements issues. For instance, if changes overlap two categories, the affected requirements may need to be refined in order to fit in one category. This may identify different evolving paths, e.g., splitting

requirements in smaller and more detailed requirements or clarify (i.e. modify) their specifications. This is the case when there are requirements overlapping software and hardware. The decision whether to allocate requirements to software or hardware may be delayed till there is a clear understanding of the system. Hence taxonomy of Requirements Evolution may classify not only an industrial context, but it can be used as a tool during design to identify requirements issues.

Requirements Evolution Measurement. Metrics [8] may be used to quantify some properties monitoring Requirements Evolution. The standard IEEE 982 [12, 13] suggests a Software Maturity Index to quantify the readiness of a software product. The Software Maturity Index can be used for software requirements, hence a Requirements Maturity Index (RMI) to quantify the readiness of requirements. Equation (1) defines the RMI³

$$RMI = \frac{R_T - R_C}{R_T} \quad (1)$$

Figure 2 shows the RMI calculated for all the software functional requirements. In this case the RMI results to be misleading to assess the readiness of the software functional requirements. The RMI does not have an increasing regular trend. Hence any assessment based only on the RMI could be misleading and risky. This result points out that it is not obvious how to apply even widely used software metrics. Metrics that are suitable at the software level can become unusable at the requirements level.

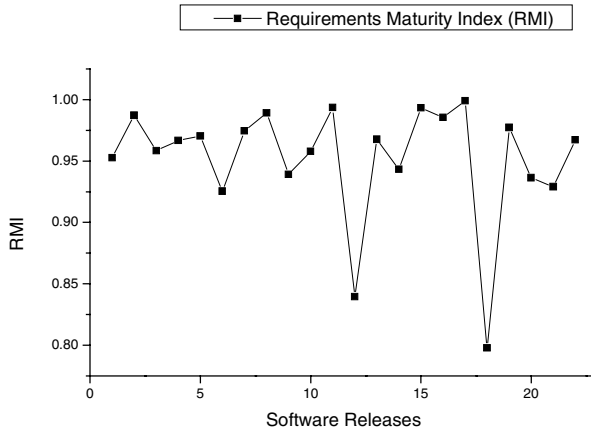


Fig. 2. Requirements Maturity Index

³ R_T quantifies the number of software requirements in the current delivery; R_C quantifies the number of software requirements in the current delivery that are added, deleted or modified from a previous delivery.

Functional Requirements Evolution. To provide a more detailed analysis our focus moves from the total number of requirements changes to the number of requirements changes in each function that forms the software functional requirements. The software functional requirements fall into 8 functions for which separate documents are maintained. Figure 3 shows the trend of the cumulative number of requirements changes for each function. The figure points out that the likelihood that changes can occur into specific functions is not constant over the software releases. An outcome of this functional analysis is that the function F1 is not likely to change, therefore, it could be considered a stable part of the system. This aspect becomes interesting, because the specific function describes the hardware architecture of the system onto which the software architecture is mapped. It seems furthermore that functions that are likely to change during early software releases change less during later releases, and vice versa. This aspect helps to relate requirements changes with the software life cycle. The different occurrences of requirements changes throughout the life cycle points out some dependencies among functional requirements. Understanding these dependencies may improve the requirements process.

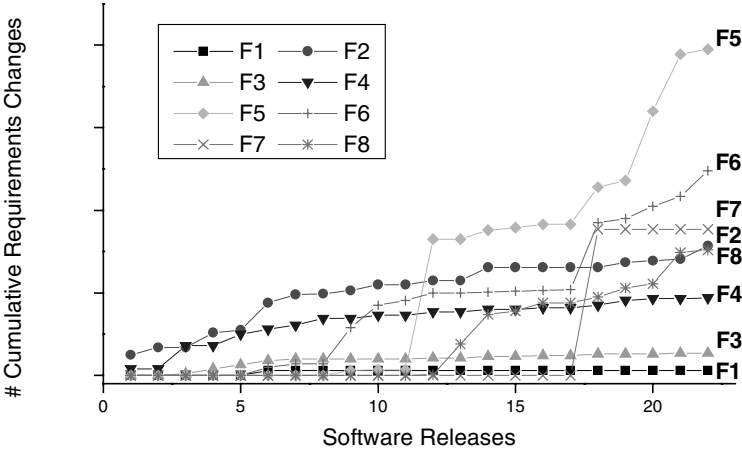


Fig. 3. Cumulative number of requirements changes for each function

Requirements Dependencies. The analysis of Requirements Evolution per software function points out some dependencies among functions. We have evaluated the dependencies among software functions by the number of common fault reports arose during the software life cycle. The number of fault reports overlapping pairwise functional requirements has quantified their dependencies. The underlying hypothesis implies that if two functions are modified due to the same fault report, then there are some dependencies between them. Table 2 shows the dependencies matrix that has been obtained according the above assumption.

We take the number of fault reports in common between two functions as *Dependency Index* between the corresponding requirements. For example, the *Dependency Index* between the functions F4 and F8 is 5, which means that there have been 5 fault reports in common between F4 and F8. The blank entries in Tab. 2 mean

that there are not common fault reports between the corresponding crossing functions. The requirements dependencies matrix, Table 2, gives a practical tool to assess to which extent software functional requirements depend each other. Moreover it identifies those particular fault reports that trigger changes into different functions. Thus an analysis of such identified fault reports may give important information about requirements. In order to provide feedback into the software organisation the matrix may be analysed to assess the ability of the organisation in identifying software functional requirements for a system or a series of systems for a particular product-line. A tool as the dependencies matrix supports software product-line engineering [26]. Future refinements over product-line projects may identify an effective set of modular system functions such that to reduce disturbing dependencies (e.g., Dependency Index equals to 1). Moreover the Dependency Index may be used to refine impact changes estimates based on traceability.

Table 2. Requirements dependencies matrix

F1	F1							
F2	2	F2						
F3		3	F3					
F4	3	1	1	F4				
F5	1	4	2	6	F5			
F6				1	1	F6		
F7				1	1	1	F7	
F8	1	4	3	5	9	2		F8

4.2 Smart Card Case Study

The analysis of the smart card case study focuses on requirements viewpoints [22]. Viewpoints provide different perspectives to manage Requirements Evolution. The case study provides the opportunity to identify hierarchical viewpoints within the organisation to which correspond different levels of management and different requirements. Viewpoints analysis furthermore points out both process and product divergences, which can be used for further investigations. The investigation analysis is based on interviews and questionnaires [3].

Requirements Evolution Viewpoints. The analysis of the smart card organisation points out three different viewpoints named *Business*, *Process* and *Product Viewpoints*. They correspond to different management levels and responsibilities within the organisation. Each level corresponds to different processes and different requirements. All the three viewpoints together contribute to the overall management of Requirements Evolution.

The business viewpoint is associated to a high management level within the organisation. This level is where projects are originated and the interaction between the organisation and customers takes part. Customer requirements are integrated together with general requirements to complete the smart card system requirements.

Requirements are then negotiated between the customer and the bureau (i.e., the department responsible for the spin-off of smart card projects). It provides production constraints (i.e., additional requirements). When customer and bureau agree on the system requirements the project is declared LIVE, that is, from this moment onward the production of the smart cards starts. The production is organised in terms of subsequent deliveries. During the production of each delivery new requirements may arise due to feedback from users of the new smart cards (e.g., misbehaves, request of new services, etc.) or to business constraints (e.g., production issues, request of additional cards, etc.). The business viewpoint therefore deals with the management of system and business requirements, which are not directly related to software requirements.

The process viewpoint consists of all the management processes adopted within the organisation [6]. Every time a request of change arises the process for changes management starts. The initial part of the changes management process is a macro-process of the negotiation activity. If changes require some software development a set of analyses is performed. Each of these analyses corresponds to a different subsystem consisting of a part of the whole smart card system. For each subsystem an impact report is produced estimating also the cost of changes in terms of man-day. The set of impact reports serves as basis for the negotiation of changes. The agreed software changes are given as input to the software development life cycle, which represents the gate to another viewpoint, namely Product, in the organisation.

The product viewpoint identifies the level producing software embedded in the smart card system. Here the software follows its own development process (i.e., a V model [17]). Software requirements are elaborated through the process and requirements changes are allocated to subsequent releases of the software behind the smart card system. At this level software changes are drawn down through the development process. Differently from the Process viewpoint that estimates changes in terms of man-day, the Product viewpoint takes into account software changes in terms of activities (e.g., coding and testing).

Hierarchical Requirements Evolution. The three viewpoints identified within the organisation represent a hierarchy through Requirements Evolution propagates. This hierarchy identifies different types of requirements and their granularity. Despite this hierarchical structure current methodologies in requirements engineering flatly capture requirements viewpoints [22].

The interviews point out to seek different abilities to support different viewpoints. Different responsibilities seek different types of support. The Business viewpoint needs to support project visibility. Most of the process-oriented methodologies in Software Engineering allow better to plan project activities, but they do not completely clarify the link between software features and project activities. This motivates a shift from process to product-oriented software engineering [26]. The Process viewpoint seeks support to enhance its management ability by measuring (requirements) evolution. The management process registers requirements changes, but a quantitative approach to measure Requirements Evolution needs to be identified within the specific environment.

The Product viewpoint would like to enhance its ability in identifying reusable (product-line) functions and repeatable processes to allocate functions to smart card system requirements. At this level there exist two different opponent processes. Good practice in Software Engineering addresses that requirements are divided into smaller

and more manageable items. This triggers an information flow expansion throughout the development process. On the other hand specific functions would be allocated to software requirements according also to past experience. The gap between these two processes represents the extent to which an organisation is able to identify an optimal and effective set of software functions. The smaller the gap, the better the ability in reusing software functions and identifying product-line features.

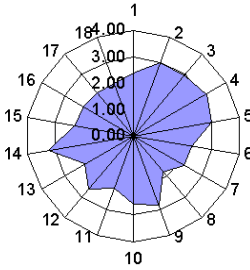
Viewpoints Analysis. The empirical analysis also investigates Requirements Engineering viewpoints by a questionnaire [3] to assess the general understanding of the requirements process within the organisation. The questionnaire consists of 152 questions grouped according the categories in Fig. 4.

- | | |
|---|---------------------------------|
| 1. Requirements Management Compliance | 10. Requirements Description |
| 2. Business Tolerance Requirements | 11. System Modelling |
| 3. Business Performance Requirements | 12. Functional Requirements |
| 4. Requirements Elicitation | 13. Non Functional Requirements |
| 5. Requirements Analysis Negotiation | 14. Portability Requirements |
| 6. Requirements Validation | 15. System Interface |
| 7. Requirements Management | 16. Requirements Viewpoints |
| 8. Requirements Evolution & Maintenance | 17. Product-Line Requirements |
| 9. Requirements Process Deliverables | 18. Failure Impact Requirements |

Fig. 4. The groups of requirements engineering questions

People with different responsibilities within the organisation filled in the questionnaire. Figure 5 shows the profiles of the questionnaire for two persons with similar experience and with different responsibilities within the organisation. They correspond to two different management levels within the organization and are respectively associated to the Product and Process viewpoints. The questionnaire captures how the requirements process is interpreted from different viewpoints within the organisation.

SW Development Manager



Project Manager I

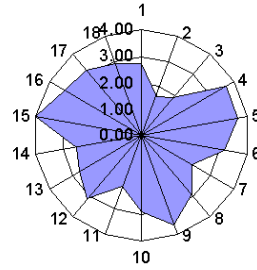


Fig. 5. Comparison of two different requirements engineering viewpoints

Figure 6 compares two similar viewpoints (i.e., the viewpoint associated to the project manager) with a third viewpoint (i.e., the viewpoint associated to the software development manager) to take into account some bias in the analysis. The similar viewpoints have similar trends, whereas there are some divergences between the different viewpoints. The largest ones are those associated with the groups of questions 2, 3, 8, 15, 16 and 17 (see Fig. 4). The group 2 and 3 identify business aspects of the requirements process. The distance between the answers to the questions in the group 8 points out that there are different levels of confidence in the management of requirement evolution. This is probably because the management of changes takes into account process aspects that better fit the process viewpoint. This is also due to some issues in transmitting requirements changes through the management hierarchy within the organisation. The groups 15, 16 and 17 identify product-oriented questions. The divergences might be due to the fact that the two viewpoints deal with different requirements. The process viewpoint deals with system requirements, which are different from the software requirements taken into account at the product level. The software embedded in a smart card system represent one aspect, which is not completely visible at the process level. These divergences identify product-oriented refinements for further investigations. They furthermore represent awareness for the organisation. Issues arising from these divergences may cause undependable software and process degradation.

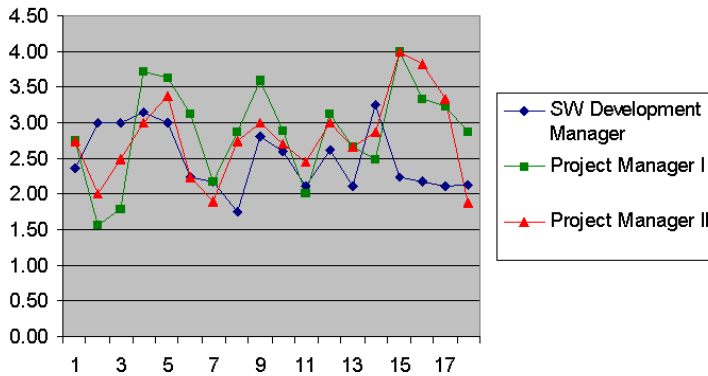


Fig. 6. Comparison of viewpoints

5 Conclusions and Further Work

This paper shows two empirical analyses of two industrial case studies respectively. Our incremental research methodology aims to identify product-oriented empirical analyses. The complexity of Requirements Evolution does not allow adopting general techniques or methodologies, moreover current practice in Requirements Engineering has not yet fully registered a shift from process to product-oriented approaches. This work tackles Requirements Evolution from a product point of view.

The analysis of the avionics case study points out product evolutionary features that improve our understanding of Requirements Evolution. The analysis gathers information starting from a general analysis of requirements evolution towards a product-oriented analysis via taxonomy of Requirements Evolution and measurement. The analysis furthermore identifies general practical issues and remarks that may be investigated in other industrial contexts. The analysis of the smart card case study identifies viewpoints within the organisation. These viewpoints represent not only sources of requirements and evolutions, but also a management hierarchy for Requirements Evolution. Viewpoints analysis may also identify inconsistencies in the requirements process and areas for further investigations. Figure 7 shows the steps of the two analyses. The joining of the two analyses identifies an Empirical Framework to analyse Requirements Evolution within industrial contexts. The framework consists of analyses that can be repeated within industrial contexts with affordable costs.

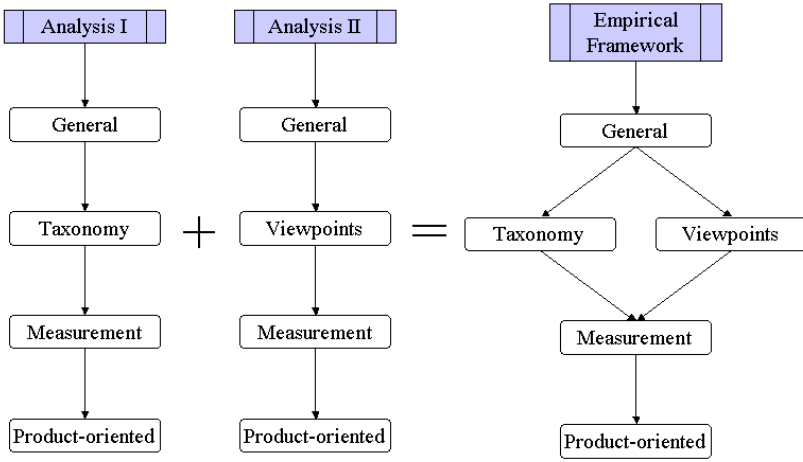


Fig. 7. Empirical framework for Requirements Evolution

Further work aims to validate the entire framework in other empirical investigations of industrial case studies. The empirical framework can then be used to analyse industrial contexts to enhance our ability in managing Requirements Evolution. On the other hand the framework can be reversed, in the sense that it can identify an effective way to structuring Requirements Evolution. This aspect will be assessed by further empirical analysis also supported by formal modelling.

In conclusion, this work identifies an empirical framework for the analysis of Requirements Evolution. The framework represents a valuable tool to implement a feedback into software process and product. Such feedback is the basis for a continuous software process improvement. The empirical nature of the work allows industry to benefit of our experience and studies.

Acknowledgements. We thank the industrial partners, who provided the case studies. Due to the confidentiality agreement with the industrial partners we cannot provide more detailed information. Despite this the work still remains valuable and the results are clearly expressed. This work has been conducted within the DIRC project [7]. The DIRC project has been funded by the UK EPSRC (Engineering and Physical Sciences Research Council).

References

1. Stuart Anderson and Massimo Felici. Controlling requirements evolution: An avionics case study. In Proceedings of SAFECOMP 2000, 19th International Conference on Computer Safety, Reliability and Security, LNCS 1943, pages 361-370, Rotterdam, The Netherlands, October 2000. Springer-Verlag.

2. Stuart Anderson and Massimo Felici. Requirements changes risk/cost analyses: An avionics case study. In M.P. Cottam, D.W. Harvey, R.P. Pape, and J. Tait, editors, Foresight and Precaution, Proceedings of ESREL 2000, SARS and SRA-EUROPE Annual Conference, volume 2, pages 921-925, Edinburgh, Scotland, United Kingdom, May 2000.
3. Stuart Anderson and Massimo Felici. Requirements engineering questionnaire, version 1.0, January 2001.
4. Barry W. Boehm. Software Engineering Economics. Prentice-Hall, 1981.
5. Barry W. Boehm. Software engineering economics. IEEE Transaction on Software Engineering, 10(1):4-21, January 1984.
6. CCTA, editor. Prince2 Manual - Managing successful projects with PRINCE 2. CCTA - Central Computer and Telecommunications Agency, 1998.
7. DIRC Project. Interdisciplinary research collaboration in dependability of computer-based systems. <http://www.dirc.org.uk/>.
8. Norman E. Fenton and Shari Lawrence Pfleeger. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, second edition, 1996.
9. Tom Gilb and Dorothy Graham. Software Inspection. Addison-Wesley, 1993.
10. Theodore F. Hammer, Leonore L. Huffman, and Linda H. Rosenberg. Doing requirements right the first time. CROSSTALK The Journal of Defense Software Engineering, pages 20-25, December 1998.
11. Ivy F. Hooks and Kristin A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. AMACOM, 2001.
12. IEEE. IEEE Std 982.1 - IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.
13. IEEE. IEEE Std 982.2 - IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.
14. M.M. Lehman. Software's future: Managing evolution. IEEE Software, pages 40-44, Jan-Feb 1998.
15. Nancy G. Leveson. SAFeware: System Safety and Computers. Addison-Wesley, 1995.
16. Mark C. Paulk et al. Key practices of the capability maturity model, version 1.1. Technical Report CMU/SEI-93-025, Software Engineering Institute, Carnegie Mellon University, February 1993.
17. Shari Lawrence Pfleeger. Software Engineering: Theory and Practice. Prentice-Hall, 1998.
18. PROTEUS Project. Meeting the challenge of changing requirements. Deliverable 1.3, Centre for Software Reliability, University of Newcastle upon Tyne, June 1996.
19. James Robertson and Suzanne Robertson. Volere: Requirements specification template. Technical Report Edition 6.1, Atlantic Systems Guild, 2000.
20. Suzanne Robertson and James Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.
21. RTCA. DO-178B Software Considerations in Airborne Systems and Equipment Certification, 1992.
22. Ian Sommerville and Pete Sawyer. Requirements Engineering: A Good Practice Guide. John Wiley & Sons, 1997.
23. George Stark, Al Skillicorn, and Ryan Ameele. An examination of the effects of requirements changes on software releases. CROSSTALK The Journal of Defense Software Engineering, pages 11-16, December 1998.
24. Neil Storey. Safety-Critical Computer Systems. Addison-Wesley, 1996.
25. Axel van Lamsweerde. Requirements engineering in the year 00: A research perspective. In Proceedings of the 2000 International Conference on Software Engineering (ICSE'2000), pages 5-19, Limerick, Ireland, June 2000.
26. David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.
27. Karl Eugene Wiegers. Software Requirements. Microsoft Press, 1999.

A Case Study on Scenario-Based Process Flexibility Assessment for Risk Reduction

Josef Nedstam¹, Martin Höst¹, Björn Regnell¹, and Jennie Nilsson²

¹Department of Communication Systems, Lund Institute of Technology, Box 118,
SE-221 00 Lund, Sweden
{josefn, martin, bjornr}@telecom.lth.se

²Ericsson Mobile Communications AB, SE-221 83 Lund, Sweden
jennie.nilsson@ecs.ericsson.se

Abstract. Flexibility is a desired quality of software processes. Process flexibility implies a capability to adapt to new contexts. Another aspect of flexibility is the cost of maintaining process effectiveness as new situations arise. A lack of preparedness for future events may constitute a high risk to a software development organization. This paper presents a method for assessing the flexibility of an organization and its processes. The assessment method is scenario-based and provides an estimate of process flexibility in terms of risk. The method is evaluated in a case study, where the process flexibility at a telecommunication software developer has been assessed. The case study indicates that the method is feasible and effective, and that the cost of conducting scenario-based process flexibility assessment is reasonable. The proposed method was able to identify a number of relevant areas to be improved in order to reduce risks of inflexibility for the particular process.

1 Introduction

Software processes are intended to be used in many projects under a number of various circumstances. In many cases, experience and knowledge can be transferred from project to project in terms of common models of the process [1]. This would not be possible if the process was not reusable and adaptable. In [2], process adaptability is classified into, for example, suitability to support varying sizes of projects, suitability for different types of products, and flexibility to accommodate a variety of methods, techniques and tools. Process maintainability, which is related to adaptability, is discussed in [3] in terms of the process' ability to evolve in order to reflect changing organizational requirements or identified process improvement proposals. A number of similar definitions of flexibility in general and process flexibility in particular are discussed in [4].

This paper focuses on the flexibility of the process in terms of its ability to maintain support for software development when changes in the environment occur. The changes may or may not lead to that the process must be adapted. Examples of changes are that the organization enters a new market or that the resources available to the organization changes.

There are similarities between the usage of processes and the usage of software architectures. An architecture may be used in a number of projects, and it may be adapted for in different situations. It also represents experience and knowledge that is applicable in a series of projects. In this paper an assessment method, SAAM [5], which is used for assessing architectures, is used as a basis for an assessment method for software processes. The derived method is evaluated in a case study in an industrial setting.

The purpose of the proposed assessment method is to determine the flexibility of a software organization and its process. The flexibility may be measured in terms of the cost of maintaining the same degree of support when the process is used in a changed setting, or when it is adapted to fit a changed setting. This cost, combined with the probability of experiencing this changed setting, can be interpreted as a risk, and the assessment method proposes areas to focus on in order to reduce process-related risks.

The structure of the paper is as follows. Chapter 2 discusses related work and the theories the paper is based on. Chapter 3 describes the proposed assessment method, built from the theory discussed in Chapter 2 and a case study. That case study, performed at Ericsson Mobile Communications, is presented in Chapter 4. Chapter 5 discusses the conclusions from the case study, and issues of further research.

2 Background and Related Work

The basic ideas of the proposed method are based on the Software Architecture Analysis Method (SAAM) [5]. The main contribution from SAAM is the usage of scenarios and a framework for developing and analyzing these scenarios. The scenario approach was selected as scenarios have been proven useful as tools for change management and decision support [6]. In [4] some examples of how flexibility can be decomposed in terms of structural and process flexibility are given. This work has been a basis for decomposing process flexibility in terms of organizational factors, decompositions used throughout this article.

SAAM primarily investigates the modifiability of a software architecture, an attribute closely related to flexibility. SAAM is performed in the following way:

1. **Develop Scenarios:** A list of expected uses or changes to the analyzed architecture is produced. It is important to consider all stakeholders affected by the architecture. The scenarios are also organized to indicate relations among them.
2. **Describe Candidate Architecture:** The architecture is described so it can be analyzed. This step is done in parallel with the previous, as need for detailed architectural information emerges when new scenarios are found.
3. **Classify Scenarios:** The scenarios are classified as direct or indirect with respect to the architecture. A direct scenario is supported by the architecture without alterations. An indirect scenario requires some changes to the architecture in order to be supported.
4. **Perform Scenario Evaluation:** In this step every indirect scenario is analyzed to find out the effort or cost needed to support it.

5. **Reveal Scenario Interaction:** If many scenarios require changes to the same component, that component probably has a high structural complexity. Such scenarios are said to interact.
6. **Overall Evaluation:** The cost of using the architecture is calculated by assessing the cost for each scenario and scenario interaction.

To translate the principles of SAAM from architecture modifiability to process flexibility, practical guidelines to evaluation [7, 8] has been used. Inspiration for this approach to create an evaluation has come from [9].

3 The Assessment Method

This section describes the assessment method, as it was used in the case study that is presented in the following section. An overview of the method is given in Fig. 1. The method consists of five activities, which are approximately carried out in sequence. First the assessment is planned. After the method's planning phase, the assessed process is described. This description matures throughout the whole assessment, and is an aid for selecting relevant interviewees. The assessor then develops scenarios from a list of issues that emerge during interviews. After an initial categorization of these scenarios the participants assess the probability and cost of each scenario, from which a risk can be calculated. The scenarios with the highest risks are analyzed to provide decision support and recommendations for process improvement. These five steps are described below.

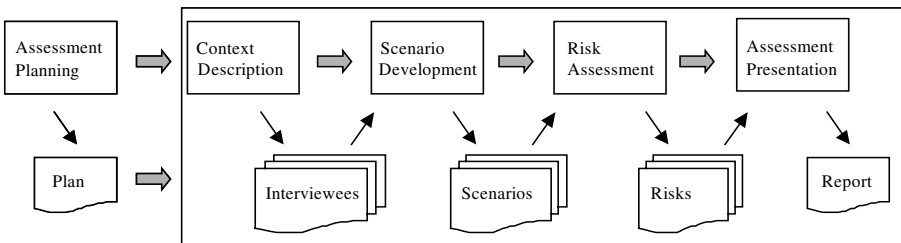


Fig. 1. Assessment method overview

3.1 Assessment Planning

The purpose of this phase is to set the constraints of the assessment and find relevant interviewees. The only input to this activity is the client of the assessment. The assessor gives the client an outline of the assessment plan for resource estimation, and a description of what the assessment will result in. Thereafter the assessor and the client agree upon a purpose, a scope, a timeframe, and a system of metrics for the assessment.

The purpose is used to focus the assessment, and the scope is used to further narrow the assessment to e.g. a particular type of subproject. The timeframe is used to narrow the applicability of the assessment in time, in order to ask more concrete

questions. The system of metrics is used to collect data from the interviewees about the cost of supporting the scenarios that will be developed, and to determine the probability of these scenarios. The cost of supporting a scenario is regarded as the cost of maintaining the same degree of support although the process environment changes due to that scenario.

Finally, interviewees for the assessment are selected in order to be able to plan the assessment. The context description developed in the following subsection is used to identify the roles in the organization that would give the most valuable information. Therefore, these two activities are carried out in parallel. The interviewees shall cover different aspects of the process and organization, limited by the scope.

3.2 Context Description

The purpose of the context description is to identify roles in the organization that are suitable for interviews, and to create a starting point for open-ended interview questions.

In order to do this a process description, an organizational structure description, and a list of roles and their responsibilities shall be created. The vocabulary of the organization can also be useful, in order to simplify the interviews.

3.3 Scenario Development

The purpose of this phase is to develop a set of scenarios that is the basis for the risk assessment. The input to this phase is a set of interviewees and an initial knowledge about the process and organization. The assessor develops scenarios from *issues* discussed with the interviewees.

The interviews are based on the initial knowledge of the process and the purpose of the interviews is to elicit and discuss issues that affect the current situation at the organization, see Fig. 2. The activities of the process and the different parts of the organization can be used as an initial set of such issues. The set of issues is expanded during the interviews.

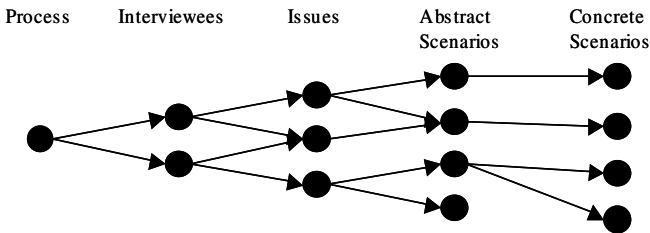


Fig. 2. Scenario development: The context description provides suitable interviewees. The outcome of the interviews is a set of issues. Abstract scenarios are elicited from the issues. These are, if possible, specified to more concrete scenarios

Ideally the interviewees provide the assessor with scenarios, but most likely the assessor will have to create scenarios from the issues discussed. Analyzing how the

issues were brought up provides clues as to how they are related. From the issues a set of *abstract scenarios* is sketched. Each issue shall be covered, but several issues may affect an abstract scenario due to their relations. When the issues are covered the abstract scenarios are specified, resulting in *concrete scenarios*, see Fig. 2. An abstract scenario may then result in several concrete scenarios. Concrete scenarios shall not contain too long chains of events, as it becomes more unlikely that the organization would take actions according to that chain of event. In such a case it may be possible to split the scenario. Some abstract scenarios may be hard to specify, due to lack of domain knowledge. If these abstract scenarios are included they introduce a higher risk of interpretational errors.

The set of scenarios shall then be reviewed in order to see if it is possible to answer whether the scenario is likely, costly, and which of the factors mentioned above that the scenario affects.

3.4 Risk Assessment

The purpose of this phase is to assess the risks of the scenarios and draw conclusions. The input to this phase is the set of scenarios and the set of issues from which the scenarios were developed.

A second round of individual interviews lets the interviewees state the probability of each scenario and the cost of supporting each scenario according to the scales determined in the assessment planning phase. The participants shall be given the option not to answer a question if they feel they have no reasonable answer. The method is however designed to only give an indication of the process-related risks that the organization faces so no exact answers are required.

The interviewees shall also state which *engineering environment factors* the scenarios affect, in order to simplify scenario interpretation. A suggested set of factors is: Process, Resources, Organization, Competence, Software Architecture, Technology, Tools, Process Implementation and Management. This set of factors is not firmly defined and many have to be reinterpreted and adapted to the specific situation. The purpose of it is to be able to draw conclusions that are more general than those who only concern a specific scenario.

After the second round of interviews each scenario can be given a risk value. This can be done in a number of ways. The basic principle is to multiply the cost of a scenario with the probability of a scenario. This is exemplified in the case study.

If the interviewees disagree to a large extent, the calculated risk values may be misleading. More than one technique of calculating them can then be used to determine an appropriate set of high-risk scenarios. The level of agreement per scenario can be analyzed with the dispersion of the risk values, which can be calculated as the standard deviation divided by the mean risk for that scenario [10].

The subjects' level of agreement can be investigated with a Kappa analysis [11]. It compares the answers given by each possible pair of interviewees. The analysis method requires that the answers are divided in a few discrete steps, which puts constraints on the selected system of metrics. This means that the method produces two values of agreement for each pair of interviewees, one for the costs and one for the probability of all scenarios.

With a Kappa analysis a matrix is built, where each cell (x,y) contains the number of times interviewee one gave an answer x when interviewee two gave an answer y . Elements on the diagonal contribute to a higher Kappa index, indicating that the participants agreed completely, and results that are close to the diagonal can contribute according to a weight given by Fleiss and Cohen [11]. A Kappa of above 0.21 is considered 'fair', above 0.41 'moderate', and above 0.61 'substantial', according to a 'rule of thumb' suggested by Landis and Koch [11].

The proposed assessment method has now produced an appropriate number of high-risk scenarios that the organization could focus on in future process improvement initiatives. This set of scenarios is however very specific, but can be generalized by analyzing the interview issues the scenarios originated from.

Another generalization that can be made is to study the engineering environment factors that the high-risk scenarios affect. For each scenario the participants should have stated which engineering environment factors are most affected. These can be analyzed by counting how many times each factor was mentioned by the interviewees and weighting this against the risk value of the scenario in question. Dividing this by the total risk value of all scenarios gives a set of indices between zero and one. These indices suggest how much each factor affects the total risk on the organization, and shows which factors should be prioritized in risk-reduction efforts.

The output of this phase is a set of high-risk scenarios, related to a set of organizational issues and a set of prioritized engineering environment factors. If the level of agreement between the interviewees has been analyzed, information about e.g. communication flow can emerge as a side effect.

3.5 Assessment Presentation

The purpose of this phase is to present the outcomes of the previous phase to the clients and the participants of the assessment in an understandable way. It is important to let everybody that has dedicated time and effort on the assessment give and get feedback. This feedback is part of the presentation phase, and it can be valuable to strengthen the conclusions from the assessment. The results may furthermore have to be presented in different versions for various audiences.

4 Case Study

An initial evaluation of the method was performed on a software process used in a software engineering course including an industry-like project [12]. The purpose was to evolve the procedures of the method. The results were guidelines for the interviews of the proposed method. In order to evaluate the resulting method a case study was planned and performed at Ericsson Mobile Communications (ECS) in Lund, Sweden.

4.1 Context Description

The application platform process used at ECS was assessed in the case study. The purpose of a platform is to serve as a basis for several products released within a time-period. The products can be developed in parallel, and the first product to be based on a platform is developed simultaneously as the platform. The first product therefore has a large impact on the requirements of the entire platform. A platform contains 90 - 95% of the code of a product, and most of the code in a platform is reused from previous platforms.

The project studied in this assessment was the second platform project to result in a product for the public market, and therefore the details and interactions of the platform and product processes were still taking form. The project was selected because it was in its requirement specification phase, the part of the process that is most mature.

During this phase the market requirements are first analyzed and refined to requirements for the various user functions. They are then decomposed into module requirements. The step from functional to module requirements involves design work, as the requirements are mapped onto an existing architecture. The work during the requirement phase is organized in 15 functional groups, each responsible for a well-defined functional area. Those groups include the roles responsible for product management, requirement specification and software design and their task is to analyze, document and prioritize the requirements and also to make sure that there is compliance between the market requirements and the functional requirements. When the requirements are put in baseline, the project is reorganized into modules that shall implement the functions. During this transition an integration plan is made to determine which functionality each module shall contain at the end of each development increment. After implementation the modules are tested and integrated, while the integration- and system test is currently the responsibility of the products that use the platform.

4.2 Evaluation Planning and Operation

The process was studied resulting in a context description, and the assessment was planned with the Process Developer at ECS. The assessor and the Process Developer decided on a timeframe including the current and the following platform project. Table 1 shows the system of metrics for data collection that was selected.

Table 1. Estimate scales

Value	Scenario Probability	Scenario Cost
1	Never	Already supported
2	Unlikely	Inexpensive
3	Likely	Expensive
4	In all likeliness	Impossible

Four roles were selected as interviewees, to cover as much as possible of the requirements process. The Process Developer gave a presentation of the process and organization, which was the base for the context description. From this description the Process Developer, the Project Manager, a representative of the System Designers and

a Subproject Manager responsible for a module with new functionality were selected as interviewees.

In the first set of interviews a set of issues was developed. The first interview was held with the Process Developer, and it was based on a set of generic questions regarding the roles and responsibilities of the interviewee, and a small set of issues found when describing the process. The interviews with the other three participants followed the same basic procedure, but were supported by a larger set of issues. The interviews were recorded on tape in order to easier find abstract scenarios during the following analysis of the issues. The interviews lasted less than one and a half hour, and apart from the assessor and the interviewee, the Process Developer was also present at all interviews.

The interviews resulted in 26 issues, listed in Table 2. These were ordered in an Entity-Relation diagram and all that had been said regarding each issue was assembled under each issue.

Table 2. Interview issues

Reuse	Architecture	Platform Vision	Communication
Platform vs. Product	Architectural Changes	Requirements	Technical Evolution
Integration	Testing	Quality	Subcontractors
Resources	Line vs. Project	Market	Organization
Process	Estimates	Tools	Competence
Decision Support	Requirement	Metrics	Organizational
	Changes	Requirement	Culture
Business Focus	Responsibilities	Sources	

This led to a set of abstract scenarios for each issue. Some of these covered several issues, and they were all traceable back to their original issues. It was realized that many of these scenarios would be hard for the interviewees to assess. One of the strengths of scenarios is to make an abstract phenomenon more concrete. A risk is however that the scenarios become too narrow, and that closely related scenarios do not result in similar values for probability and cost. Therefore both abstract and concrete scenarios were provided during the second interview, where the concrete scenario acted as an example among others within that abstract scenario. Some abstract scenarios were however found to be too broad and had to be split up into several concrete scenarios. Every scenario was not made concrete due to lack of domain knowledge. The result was a set of 36 abstract scenarios, whereof 31 were exemplified with concrete scenarios. Table 3 shows some of these scenarios.

During the second set of interviews the participants were asked to assign values for probability and cost of each scenario according to the scales above, and determine which engineering environment factors where affected by the scenario. The original list of factors was process, organization, competence, resources, architecture and technology. It was derived from the issues and determined by the assessor and the Process Developer. During the first and second interview the list was appended with three additional factors that emerged during the interviews: tools, process implementation and management.

During the first interview, which was held with the Process Developer, interpretation problems were found with three scenarios. These were clarified and caused no problem. During the second interview a scenario was however found that had to be split in two. The scenario concerned architecture and the System Designer realized that two similar problems called for different solutions. This was confirmed by the two remaining interviews.

The Process Developer did not attend the other interviews during the second round of interviews. The interviews were recorded on questionnaires, and they took less than an hour each.

Table 3. Sample scenarios

Abstract Scenario	Concrete Scenario
Applications must be rebuilt after an architectural change.	The architecture must be rebuilt due to new performance requirements, which means that the interfaces to most applications must be rebuilt.
Problems during integration of the second product based on the current platform.	Performance bottlenecks do not emerge until building the second product on this platform.
Products set quality requirements on the platforms.	The first product to be built on this platform demands that X% of all branches in code shall have been run in test.
Project managers get a larger resource responsibility.	The line organization functions as a consultancy firm for the project organizations.

4.3 Evaluation Data Analysis

Since the assessment method is qualitative in nature, it is hard to draw significant quantitative conclusions from the results. The assessment is intended as a tool for decision support for future process improvements, indicating problem areas in the process and organization.

Since the interviewees gave intermediate values during the second round of interviews, the scales found in Section 4.2 were modified to include seven steps; the four original and three intermediate steps. These steps were numbered from 0 to 6, as it was natural for both the probability and the cost scale to start at zero.

Risk Assessment. The risk values for each scenario were calculated by assuming that the original values for cost, c_{is} , and probability, p_{is} , that an interviewee i gave for a scenario s were dependent. The values were assumed to be dependent, because when the interviewees studied a scenario they in most cases automatically assessed the risk in some way. This was noted by frequent phrases such as “we will not be able to support this scenario but it doesn’t matter, because it will never happen” or “we are already working this way so the scenario has already happened and it will cost nothing to support”. Furthermore the scale for cost was assumed to be interpreted as an exponential scale.

The second assumption lead to a transformation of the cost scale from the initial range of 0 to 6, to scores ranging from 1 to 10^6 , giving the cost c'_{si} for one scenario and one interviewee:

$$c'_{is} = 10^{c_{is}}. \quad (1)$$

The probability scale was normalized, ranging from 0 to 1, giving the probability p'_{is} for one scenario and one interviewee:

$$p'_{is} = p_{is} / 6. \quad (2)$$

After these transformations the first assumption was considered, and the risk value for each scenario was calculated accordingly. The risk, r_{is} , each interviewee gave each scenario was calculated by multiplying the resulting cost with the resulting probability:

$$r_{is} = c'_{is} \cdot p'_{is}. \quad (3)$$

This resulted in two to four values of risk for each scenario, depending on how many interviewees had assigned a cost and a probability value to each scenario. Finally the average risk r_s for each scenario was calculated from the individual risk values:

$$r_s = \sum_i \frac{r_{is}}{n_s}. \quad (4)$$

In Formula 4, n_s is the number of answers for scenario s .

Fig. 3 shows the scenarios sorted according to their risk calculated according to the *exponential dependent technique* described above, with the risk value on the Y-axis and the scenario number on the X-axis. The line of diamonds shows the dispersion of the risks of the scenarios, here measured as the standard deviation divided by the average risk. The four first scenarios show much higher risk than the following, which indicates that they shall be prioritized. When analyzing them more carefully it however became clear that the participants gave very varying answers, which is also indicated by the high dispersion of the two first ones.

Participant Interagreement. The high dispersion of risk values for some scenarios called for further analysis of the level of agreement between the participants. A Kappa analysis was made on the interviewees' values of cost and probability. Kappa for the costs ranged from -0.02 to 0.34 among the possible pairs of interviewees, with an average pair agreement of 0.20. Kappa for the probabilities ranged from 0.13 to 0.42 among the possible pairs of interviewees, with an average pair agreement of 0.25. The low levels of agreement can be explained by that the interviewees were selected to get as differing views on the process as possible.

Other Risk Calculation Techniques.

Since the risk values for the scenarios were uncertain, other ways of finding the most important scenarios were investigated. Firstly, the scale for the costs transformation was studied. If the original, linear values were used, the order of the scenarios differed. It was however considered reasonable that the original verbal scale was not interpreted as linear by the interviewees, but rather exponential or worse. The *linear technique* of assigning risk values was later used as a reference when evaluating the assessment method during the feedback session.

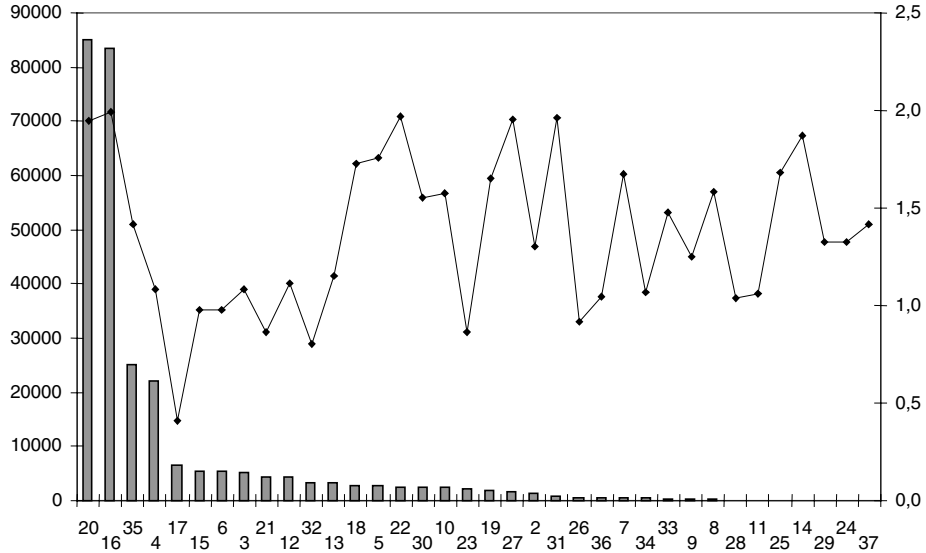


Fig. 3. The scenarios sorted by their risk calculated according to the proposed method

An analysis was made as to whether the order of the scenarios was sensitive to the base used in the original exponential transformation. The original base was ten, i.e. each of the six steps in the scale meant a tenfold increase in cost. A twofold increase was tried, and the order of the scenarios differed slightly. The four most important scenarios were however still the same. If a fourfold increase or larger was used, the ranking order was identical to the tenfold increase.

The scenarios were also sorted by selecting the answer giving the highest risk for each scenario, i.e. saying that if anyone can sense a risk in a scenario, there is a risk. This order, the *maximum-risk technique*, was somewhat different as it gave a large number of ties. The four first scenarios were however still the same as the four first of the *exponential dependent technique*. The scenarios that each individual rated highest were also examined. None of the interviewees ranked the same scenario highest and these four scenarios were identical to the four most important scenarios in the *exponential dependent technique*.

The last examined method to calculate the risks of the scenarios was to assume that the values for cost and probability of the scenarios were independent. The exponential transform was still applied to the costs for each individual answer. Then the average cost and average probability for each scenario was calculated, over the number of

answers for that scenario. These two averages were then multiplied to give a value for the risk of that scenario. This technique, the *exponential independent technique*, implies that the participants, when studying one scenario, gave the answer for cost independently of the answer for probability. This would perhaps be beneficial, but was not the way in which the answers were given. Using the *exponential independent technique*, two of the four most important scenarios were found among the four most important scenarios from the *exponential dependent technique*.

The *exponential dependent technique* was selected as the prime candidate for calculating risk in this particular case, a choice that was confirmed at the feedback session. Most of the following analysis is based on this assumption.

Analysis of Engineering Environment Factors. The nine engineering environment factors could now be analyzed based on the calculated risk values for the scenarios. The sum of the risk values for all scenarios where one of these factors had been brought up by an interviewee was calculated. This value represents the risk that affects that factor. By dividing it by the total risk for all scenarios the percentage of the organizations risk affecting that factor was calculated. This analysis is visualized in Fig. 4. ‘Implementation’ in the diagram means process implementation, while the other factors are the same as in Section 4.2. The sum of the affected factors is more than one, since several factors appear in a single scenario.

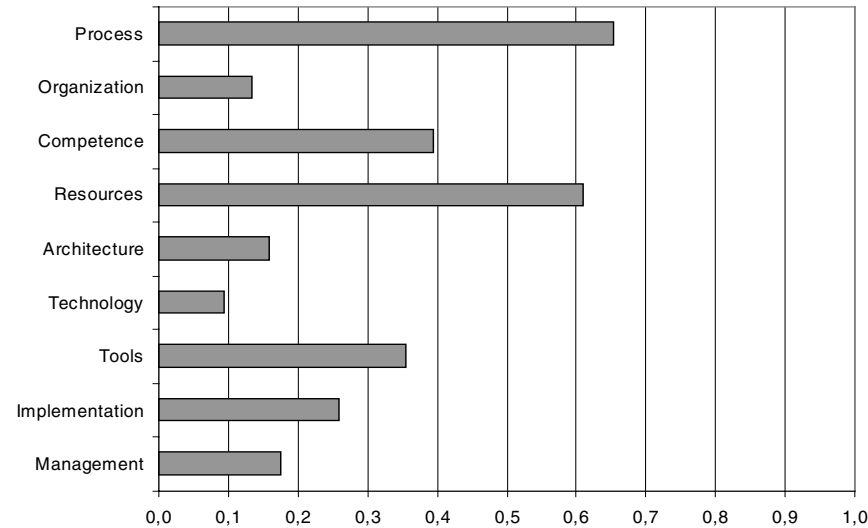


Fig. 4. A diagram showing the impact each of the engineering environment factors has on the risk of all scenarios

The figure shows that 65 % of the discovered risk somehow affects the process, and that 61 % affects the resources. Therefore it would be recommended to focus on these two factors when trying to reduce risks according to the results of this assessment.

4.4 Evaluation Results

The analysis gave clues to which scenarios were most important. Because of the low level of agreement between the interviewees, three techniques of calculating risk were considered. The techniques were: *exponential dependent*, *exponential independent* and *linear*, as discussed in Section 4.3. The four highest-risk scenarios from each technique were selected, which resulted in nine unique scenarios. These were compared to the interview issues, and three groups of scenarios emerged. These were:

- Architectural issues related to rebuilding the architecture due to e.g. new technologies, and architectural problems with system integration.
- Resources and competence, which are lacking if Ericsson has to react to the high-risk scenarios or introduce new technology.
- Process issues mostly regarding the relation between requirements and test.

Among these issues, process, resources, and to some degree competence and tools, had already been pointed out by the previous analysis of engineering environment factors. One of the nine scenarios did not fit in these groups, as it only affected the issue of subcontractors. This scenario was also regarded as least important of the nine by the three selected risk calculation techniques. These results were presented to the interviewees on a feedback meeting.

Feedback Session. A meeting was held with the participants in order to present and verify the results. An exercise was conducted in order to select the most appropriate technique for risk calculation. The participants were given three sets of high-risk scenarios, consisting of the four scenarios with highest risk according to the three techniques *exponential dependent*, *exponential independent* and *linear*. They were then given the question: If you were to start a process improvement initiative, which set of scenarios would you base it on? All four participants selected the four scenarios given by the *exponential dependent technique*. This was a verification of the assumptions made during the risk assessment, and the presentation that followed was based on the selected technique.

The results above were presented and the participants agreed to the risks associated with resources and the issue of improving the connection between their requirements phase and test phase. They showed interest in analyzing their low level of agreement, and agreed on letting the Process Developer analyze the data from that aspect. They were furthermore satisfied with the level of resources they had spent on the assessment. Together they had spent less than 30 working-hours, where the Process Developer had done the main part. A presentation of the whole case study for middle and senior management was scheduled.

Threats to Validity. Some threats emerged during the assessment operation.

The Process Developer had participated during the first set of interviews. This hindered anonymity, but the discussions were open and the other interviewees were not afraid to criticize the process or other aspects of the organization. This decreases the threat that important issues were not brought up during the interviews.

A major concern is the quality of the scenarios. A high-quality set of interview issues is a base for scenario development, as is the assessor's knowledge of the organi-

zation, found in the context description. The assessor's ability to develop an adequate set of scenarios from this material is however a decisive factor. The method gives support but training and practice is also essential. One way of validating the scenarios is to analyze future process changes in the organization, and the causes for these changes.

A threat during the second set of interviews was that all interviewees except one used intermediate values, perhaps because words like 'never' and 'impossible' are awkward to assign to events. The interviewee who did not use intermediate levels also filled in the questionnaire himself, which the others did not, and he assigned factors to scenarios after he had assigned all cost and probability values. The impact of such threats is hard to assess.

From a correct set of high-risk scenarios the identification of high-risk issues should present no threat. The generalization from high-risk scenarios to prioritizing engineering environment factors however contains a number of threats. Firstly the factors were not thoroughly defined and were left for the interviewees to interpret. Secondly the set of factors was created informally and important factors may be overlooked. These threats could be avoided with a better defined set of engineering environment factors.

5 Conclusions and Further Work

This paper has introduced the concept of process flexibility. A method is proposed which assesses process flexibility by analyzing risks within an organization and its process. The method uses scenarios in order to focus on the particular situation at an organization. The method is evaluated in a case study, where the process flexibility at Ericsson Mobile Communications has been assessed. The method has proven promising in the aspects of cost, feasibility and effectiveness:

- **Cost.** During the case study, the assessed organization provided three participants who spent less than 30 working-hours in total. Of these thirty hours the contact person at the organization spent about a third. The assessor spent around 60 hours collecting and analyzing data. The cost of performing an assessment of this type is therefore considered low.
- **Feasibility.** The low cost also leads to increased feasibility, as there were no major problems in scheduling appointments with the participants. An assessor using this method will need some training, most of which can be derived from this article. With a trained assessor the method generates questions and scenarios that are possible to answer and assess unambiguously. The results generated are also easy to interpret and feasible to act upon.
- **Effectiveness.** The method has in the case study produced results that the participants agreed described the situation at the organization. Three areas were identified that called for improvements.

The method has only been used once, and has many opportunities of improvement. From a researchers point of view, the method also has opportunities of generating general conclusions about process flexibility. To further increase the feasibility of the

method the guidelines and procedures of the method could be improved, either from experience with the method or from lessons learned from other software process assessment methods.

The effectiveness of the method could be validated by following up the situation at assessed organizations. One possible way of doing this is to see if any of the scenarios has occurred in the timeframe specified in the assessment. In order to make generalizations from this method it must first of all be possible to compare results from different assessments. If this is possible one can eventually compare organizations that are flexible, according to the method, versus organizations that are inflexible. This could provide general conclusions as to how organizations and processes should be designed in order to be flexible.

Further studies of the proposed method may provide additional insights into ways of understanding and improving scenario-based process flexibility assessment, which hopefully can contribute to an effective and efficient method for risk management in software development.

References

1. Basili, V. R., Caldiera, G., Rombach, H. D.: The Experience Factory. In: Marciniak, J. J. (Ed.): *Encyclopedia of Software Engineering*, Vol. 1. John Wiley & Sons, Inc., New York (1994) 469-476
2. Zahran, S.: *Software Process Improvement: Practical Guidelines for Business Success*. Addison-Wesley, Harlow, UK (1998)
3. Sommerville, I.: *Software Engineering*, 6th Edition. Addison-Wesley, Harlow, UK (2001)
4. Nelson, K. M., Nelson, H. J., Ghods, M.: Technology Flexibility: Conceptualization, Validation, and Measurement. In: Nunamaker, J. F. Jr., Sprague, R. H. Jr. (Eds.): *Proceedings of the 30th Hawaii International Conference on System Sciences*, Vol. 3. IEEE, Los Alamitos, California (1997) 76-87
5. Bass, L., Clements, P., Kazman, R.: *Software Architecture in Practice*. Addison-Wesley, Reading, Massachusetts (1998)
6. Jarke, M., Kurki-Suonio, R.: Guest Editorial: Introduction to the Special Issue: Scenario Management. In: Kemmerer, R. A., Ghezzi, C. (Eds.): *IEEE Transaction on Software Engineering*, Vol. 24, No. 12, December 1998, IEEE, Washington (1998) 1033-1035
7. Scriven, M.: *Evaluation Thesaurus*, 4th Edition. Sage Publications, Thousand Oaks (1991)
8. Robson, C.: *Real World Research: A Resource For Social Scientists and Practitioner-Researchers*. Blackwell Publishers Ltd., Oxford, UK (1993)
9. Ares, J., García, R., Juristo, N., López, M., Moreno, A. M.: A More Rigorous and Comprehensive Approach to Software Process Assessment. In: Perry, D. E., Schäfer, W., Tully, C. (Eds.): *Software Process: Improvement and Practice 2000*, 5. John Wiley & Sons, Ltd., UK (2000) 3-30
10. Regnell, B., Höst, M., Natt och Dag, J., Berenmark, P., Hjelm, T.: An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. In: Loucopoulos, P., Potts, C. (Eds.): *Requirements Engineering Journal 2001*, 6. Springer-Verlag Ltd., London, UK (2001) 51-62
11. Fusaro, P., El Emam, K., Smith, B.: Evaluating the Interrater Agreement of Process Capability Ratings. In: Bieman, J. (Chair): *Proceedings of the Fourth International Software Metrics Symposium*, IEEE, Washington (1997) 2-11
12. Wohlin, C., Regnell, B.: Strategies for Industrial Relevance in Software Engineering Education. In: Glass, R. L. (Ed.): *Journal of Systems and Software*, Vol. 49, No. 2-3, 125-134. Elsevier Science BV, Amsterdam, NL (1999)

Contexts of KM Based SPI: The Case of Software Design Experience Reuse

V. Seppänen¹, S. Komi-Sirviö², A. Mäntyniemi², T. Rahikkala³, and
M. Pikkarainen²

¹ University of Oulu, Department of Information Processing Science
PL3000, FIN-00014 Oulun yliopisto, Finland

Veikko.Seppanen@oulu.fi

² VTT Electronics, P.O. Box 1100, FIN-90571 Oulu, Finland
{Seija.Komi-Sirvio, Annukka.Tauriainen,
Minna.Pikkarainen}@vtt.fi

³ Cybelius Software, P.O. Box 36, FIN-02131 Espoo, Finland
Tua.Rahikkala@cybelius.com

Abstract. In this paper we will address some contextual issues related knowledge management based software process improvement (SPI). The research question emerged during the practical case study is what are the contexts of knowledge-based SPI and, furthermore, how to take them into account when planning improvements. By context we mean complex set of organisational structures, engineering and supportive processes, products, and relationships between them. We will first describe the case that provided us data for building the context model. We will then continue by illustrating and analysing the model from two viewpoints: the inner and the outer context. At the end of the paper we will draw conclusions and general remarks from the work done so far.

1 Introduction

Software Engineering (SWE) is a more complex and distributed activity than ever, suffering from many pressures that raise needs for new approaches for improvement as also discussed in [1]. During the nineties Knowledge management (KM) (e.g., [2, 3, 4, 5]) has become a topic of active research and exploitation within SWE research area too (e.g. [6, 7, 8, 9, 10]). SWE is an extremely knowledge intensive engineering task that requires special abilities and knowledge of several methods, processes, tools, application domains, etc. For this reason in the ever-changing development environment the continuous creation, distribution and use of knowledge becomes critical for success. Furthermore, unique solutions to capture and share knowledge are needed.

In this paper we will introduce a context model that we are developing to help knowledge management based SPI. The needs and ideas for the contextual model raised during a project where our goal was to help a software development organisation to improve reuse of design experience. There had been earlier attempts to improve reuse practices utilising knowledge management in this organisation that,

however, eventually failed. This raised the question why the attempts did not succeed, and what could be done to avoid the same failures again.

Based on an analysis of the previous improvement initiative and an evaluation of the current status of the organisation, we formulated a context model to be taken into account when planning KM based SPI. The context model consists of two viewpoints: the *inner context* and the *outer context*. The inner context is seen to consist of software development processes (projects) and organisations, KM processes (projects) and organisations, and the SPI initiative. The outer context consists of the product development and line organisations, processes and the environment where the software is being produced and ultimately used (Figure 1). It is usually difficult to characterise especially the outer context, because its limits can be drawn in many different ways. For example, end-users may or may not be considered as belonging to the outer context, depending on the question if they are using the software directly. In this paper we restrict our view to the outer context into one focal organisation that develops software to be embedded into electronic products.

The paper is structured as follows: in chapter 2 we introduce the case for improving design experience reuse, then we will describe and analyse the context model derived from the case study (chapter 3), and finally end up with conclusions in chapter 4.

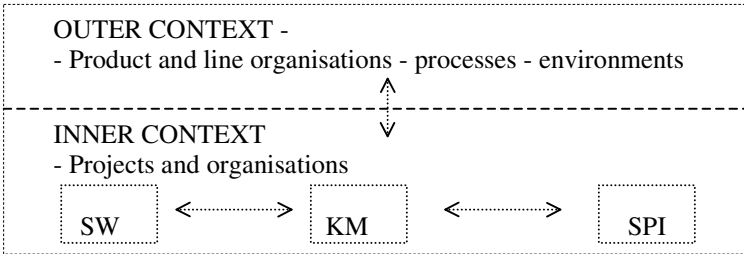


Fig. 1. The contexts of knowledge-based SPI.

2 Case - Improvement of Design Experience Reuse

The case organisation had been interested in capturing and reusing knowledge related to software design, in their own words “design experience”. The study discussed in this paper was a part of an SPI initiative aiming at improving the reuse of design experience for the needs of a particular product development project.

In this chapter we will first outline the starting point for the improvement initiative, and then present the case in question.

2.1 Starting Point for the KM Based SPI Initiative

The aim of this particular SPI action was not to create any KM system, but rather a process that would help in acquiring experience from existing sources, such as the company's databases and individual persons, for the needs on ongoing SWE projects. The focal case company is an independent business unit of a global corporation that develops electronic products. The company is organised into a matrix, where the line organisation is responsible for resource management and competence development – for example SPI, and the project organisation takes care of the product development. The specific site of the company addressed in the case study was established in 1985, and it employed hundreds of people during the time of the case study.

Both the managers and designers of the company felt that plenty of experience was wasted because it could not be found and reused, so called *Lessons to learn initiative* had been established that aimed to share useful knowledge using a common database. However, the activity log entry of the database revealed that within 590 days, 702 documents had been read, which means only some 1,2 documents per day – a small number regarding the volume of software projects. Some sections of the database included mainly outdated information from the year 1997 and the beginning of 1998. One section had been suspended. Moreover, there were many incomplete entries in the database.

It was concluded, based on an evaluation of the situation, that the approach had not been too successful – something different would be needed. One of the new ideas to test was to define a method for capturing experience not on the fly and by the software designers themselves, but by using the company's and external SPI experts as knowledge capturing agents.

2.2 Seeking for Better Practices

Capturing reusable software design experience was considered to involve the tasks of finding, acquiring, analysing and packaging of relevant pieces of knowledge, to be carried out by the SPI experts using some simple means: knowledge capturing should not force the software development and SPI organisations and processes to change "too much". This meant, in practice, that manual "off-line" knowledge capturing methods were favoured. The knowledge capturing organisation was defined as consisting of the so called *knowledge capturing projects* and *customer projects*.

This made the separation between knowledge-based SPI and software design explicit, but did not change the present organisational setting. The method was tested with one specific customer project. A template was created to structure the customer project's needs: the project members used the template to indicate what design knowledge was needed, in what form, and where they intended to reuse it.

The needs were acquired mostly in free-formatted questions structured in two ways, separating *process-related* and *product-related* knowledge. The former included items to describe e.g. software design and testing tasks, work products, roles, organisations, skills, methods, and tools. The latter included, for example, descriptions of parts of certain products and their interconnections, product family

hierarchies, etc. In order to structure the acquired knowledge, it was decided to construct *knowledge maps* that would consist of three parts: *what*, *where-general* and *where-special* design knowledge.

Special knowledge would be used only by one or two customer projects, general knowledge by several projects. The "What" part of the knowledge was based on a classification of the software design and management documents used by the company.

Opposed to Davenport and Prusak [4], organisational charts were used for building knowledge maps. Especially middle-managers shown in these charts were seen as very helpful to guide the further elaboration of the knowledge maps, because they were knowledge brokers who knew much of individual products and projects.

A process was defined to capture design experience, including three base practices: (1) definition of the scope and requirements for the knowledge, (2) acquisition of the relevant knowledge, and (3) packaging of the knowledge. The purpose of the base practice, entry criteria, other inputs, stages of the base practice, verification and validation, exit criteria, and other outputs were defined for each base practice.

The objective of the first base practice was to guide the customer project in defining the requirements for the captured knowledge, the second base practice to acquire knowledge from existing sources and store the results for further packaging, and the third base practice to package the knowledge gathered to meet the stated requirements.

3 Analysis of the Case– Identifying Contexts of KM Based SPI

At the beginning of this paper we referred to the so called outer context and inner context: the viewpoint to the problem is either from "outside" or "inside". The inner context focuses on what is inside in the organisation that is acting to solve the problem - in the case of this paper to improve the focal organisation's design experience reuse process. The inner context can be further divided into two sub-contexts: the *software* and *knowledge contexts* using process-related and organisational aspects. The outer context addresses what is outside of the focal organisation and processes. The outer context has an influence on the inner context and its alterations cause changes to the inner context, too. Unfortunately, it may be sometimes difficult to make the outer context explicit, because its limits are all but clear cut.

In this chapter we will introduce and analyse the context model (Figure 2) that was developed during the case study.

3.1 The Inner Context

The contexts of the case organisation are illustrated in Figure 2. The outer context includes the product technology organisation, resource creation process and product development organisation. The product technology organisation consists of technology groups, including the software technology group as an organisational actor. Resource creation is carried out by the line organisation to ensure appropriate

human resources for product development. The product development organisation consists of so called product development programs that include software development projects as one of their activities. The inner software context includes the software production process and software development projects, as well as the SPI organisation. The latter uses an external expert organisation in the inner knowledge context, i.e. to carry out the knowledge production process for SPI.

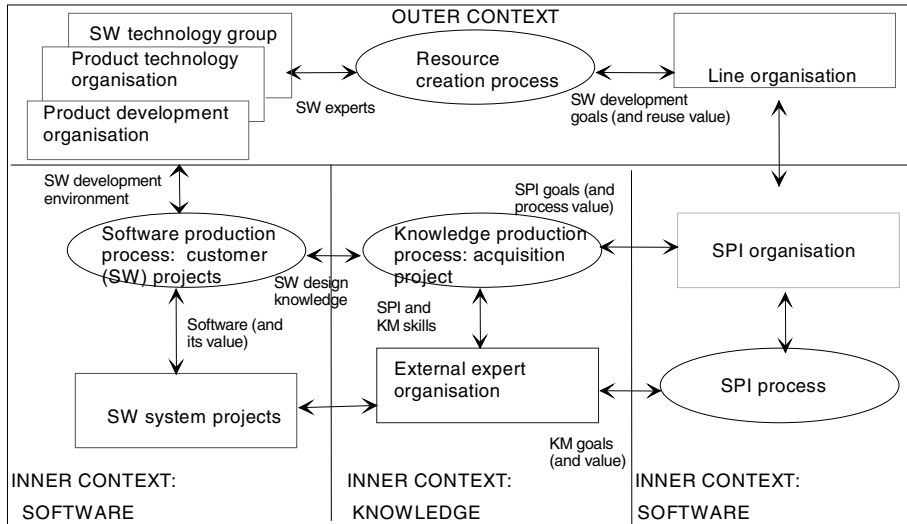


Fig. 2. The context of KM based SPI illustrated.

Although differences between the *process-related* (shown as oval in the figure) and *organisational* (shown as rectangular in the figure) elements of the context are not obvious, they exist. By the software developers, for example, SPI can be claimed to be seen more as an organisation in the case company than a process, because its main task is to create new software development processes and not so much to help to carry out software development projects. Similarly, as the line organisation is an active resource provider not only when establishing projects, but whenever resource changes are required, it can be seen more as a process than an organisation from the viewpoint of software development projects. Correspondingly, the explicit (arrows in Figure 2) and implicit (dashed arrows in Figure 2) *relationships* emphasise the process or the organisation as the basis of interaction.

Overall, the organisational and process structure of the software context shown in Figure 2 is not too lean. Both more permanent and temporary organisations were involved, such as groups and projects, one external expert party, and processes that associate everything together – most notably the resource creation process. Moreover, although the SPI process remained rather implicit, i.e. the line and SPI organisations were not related to each other by any explicit process, the knowledge production process was made explicit and intertwined with the SPI process, because the SPI organisation took care of this process.

Analysis of the Knowledge Context

Speaking of the *knowledge context*, the knowledge acquisition and management processes are taken care of by the SPI experts, carrying out the tasks of finding, acquiring, analysing and packaging the relevant pieces of design experience for reuse. The organisation, methods and tools used by this knowledge management team would be made separate from those of the software production teams.

Since the members of the former one are SPI experts, this means that knowledge-based SPI is seen much as an off-line duty from the viewpoint of software development. In other words, integration of the three parts of the inner context shown in Figure 2 was not seen as necessary in the case organisation.

The relationships between the actors involved in the case were based on somewhat different aspects. Regarding the line and SPI organisations, the relationship was mostly comprised of interaction between individuals. The same holds for the relationship of the external expert organisation and the software development organisation from which knowledge was acquired. The relationship between the expert organisation and the case company's SPI organisation involved a versatile set of joint activities, whereas the one between the product technology and development organisations was based heavily on human resource exchange. It should be noticed, however, that as the knowledge management team consisted of SPI experts, there was yet no explicit organisation for managing the knowledge context. Another way to put this is that the existing SPI organisation, a part of the software context, is also responsible for the knowledge context. It has a double role to play in KM based SPI.

The knowledge is to be acquired for customers by making a separation between *product-related* and *process-related* knowledge as well as between the viewpoints: *what*, *in which form*, and *when*. The decision to use organisational charts for structuring the knowledge sources indicates an aspect that needs once again to be emphasised: knowledge is offered *for* the customer projects, not *by* the customer projects. The latter was attempted earlier, but it failed and it was not considered as a useful knowledge structuring viewpoint. This organisation changes, too, but its structure and responsibilities are actively maintained and known by others.

Knowledge capturing is done by SPI experts for the customer projects. Therefore, it can be said that the case involves KM based SPI rather than "pure" knowledge management. Furthermore, the key role of middle managers comparable to Nonaka and Takeuchi [2] is likely to emphasise the generic know-what viewpoint, as opposed to specific areas of software development expertise of individual designers.

Key Drivers for the Inner Context – SPI Goals and Value

The case data indicates that product development goals and value related to the reuse of design expertise drove the KM based SPI effort. The explanation for this may be that autonomous project managers are strongly committed to develop products with required features and quality within given budgets and schedules. They need relevant knowledge provided for them in the right form and in the right time, they do not have the time to seek extensively for it or to provide it for others. One of the main reasons for problems encountered in the earlier approach was that the SWE personnel had been responsible for acquiring and maintaining the knowledge. Now, a kind of

internal temporary knowledge contracting scheme was outlined instead, not based on a new permanent organisational arrangement, but between much more temporary software development (knowledge customer) and SPI process (knowledge supplier) teams.

Identification of relationship between actors and organisations is important for the success of SPI, because the goals and the expected value, related to the SPI process conducted together by some actors, should be tied to the key elements of the relationship.

For example, if SPI aims at improving the software process based on knowledge captured from product development projects, there should be a direct relationship, i.e. an actor bond between the SPI organisation and the customer projects (the product development organisation). If the main SPI goal involves a more effective reuse (activity) of software design experience (knowledge resource), the relationship should emphasise process aspects, i.e. shared resources and activities. Referring to Figure 2, the relationship of the external expert organisation with the customer project was based on a temporary bond.

Although some key SPI goals could be identified, such as not changing the present software development process or organisational structures, the process value resulting from SPI nevertheless remained quite unclear in the case study. Similarly, it was not possible to make the value of KM explicit as part of SPI. This means, in practice, that KM based SPI may encounter problems even in its new form – its benefits cannot be validated, as the SPI goals and value can easily remain hidden or at least rather abstract. Analysis and management of the outer context of KM based SPI may help to avoid these problems, as will be discussed in the next section.

3.2 The Outer Context

Our view to the outer context of KM based SPI is illustrated in Figure 2. What may be even more important, however, is the *change of SPI and software development activities* in the course of time due to the evolution of the outer context. In other words, it is useful to make the change of the outer context explicit, to be able to analyse SPI goals and value creation. Referring to Figure 2, changes of the outer context involve the organisations and processes that, for example, provide the resources needed in SWE and SPI. Because SPI should result in business benefits through effective and efficient SWE, it is useful to consider the change of the outer context as evolving SPI “business cases” that can be related to SPI goals and value.

SPI Business Cases

There exist many different models for strategic change of businesses, often based on the idea of successive phases or stages. Instead of phase-driven models, where the key problem is that the phases are pre-determined, we favoured trying to make the change of SPI goals and value explicit. This resulted in four types of business cases for SPI, which we will call process repair, optimisation, re-engineering, and innovation. We will first characterise them briefly and then illustrate them in the case organisation.

Process repair means that short-term gains are sought by special SPI solutions. This is usually done by fixing the present problems of the SWE process. For example, new tools can be acquired to conduct the most critical SWE sub-process. Depending on organisational structures, process fixes may be centrally managed or distributed among different actors, such as individual software development teams or projects.

Process re-engineering has become familiar in the nineties [11, 12] as an approach to streamline processes through new organisational structures. To a large extent, SWE process re-engineering focuses on actors, revising their organisation, involvement and responsibilities. Although process re-engineering aims at general improvements in the ways things are done, the re-engineered process is usually not revolutionarily different from the past SWE process.

Process optimisation, focusing on key software development activities rather than on organisational structures, leads in general to more permanent value although it seeks for special solutions to make the SWE process more effective, and is thereby based on rather specific SPI knowledge.

Process innovation, which we consider to result in the highest and most permanent value in SPI, would mean focusing on the interplay of many processes as a whole, rather than on specific SWE process actors, activities and resources. Strategic benefits would be sought by general improvements based on generic and explicit knowledge.

Changes of SPI Business Cases in the Focal Organisation

The case discussed in this paper is an excellent example of process optimisation: the failed innovation, repair and re-engineering types of attempts to reuse past software design knowledge were replaced by a very specific approach, one knowledge capturing project (SPI experts) serving several customer projects (small teams of software designers). Figure 3 illustrates the changes (dashed arrows) of the SPI business cases in the focal organisation. The long-term value resulting from the four cases is simply estimated using the scale from very low (--) to very high (++)

It is interesting that the development obviously started from process innovation, because the earliest SPI initiative was meant to help in making better use of software design knowledge by changing the focal organisation's SWE process. The idea to capture and share software design experience systematically in software projects would have affected considerably the software development organisation. The attempts changed rather soon into the process repair type of a situation. For example, although the Lessons to learn database that had been made available as a versatile resource for design experience reuse, its most used pieces of content were final reports from past projects, used as a jump start for sketching new project plans.

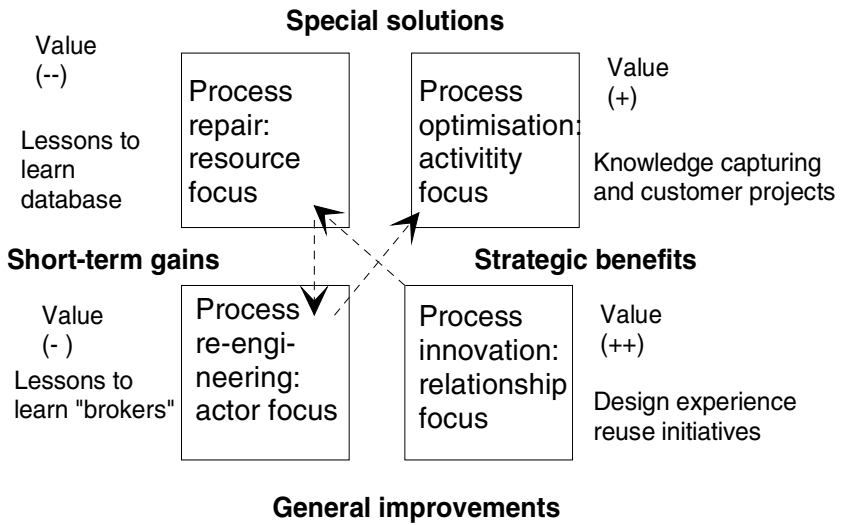


Fig. 3. Change of the outer context of the case organisation wrt. SPI business cases.

The case data shows that the use of the Lessons to learn database was tried to be made more effective by allocating new organisational responsibilities regarding the follow-up and use of the database, i.e. design experience reuse was meant to be assisted by special KM experts. There were also some information sharing events in which interested people could participate. The focus moved therefore on actors, the people who both provided knowledge to, and used the database, rather than on the database itself. In Figure 3 this is characterised as the appearance of “knowledge brokers”. However, the idea of the brokers did not work, because the software designers tended to trust on anyone nearby, rather than on the knowledge brokers – who apparently also did not have the time to carry out their special task.

Therefore, new improvements were launched, in which an external SPI expert organisation took part. This time, the focus of the improvements was on activities. The SWE process was intended to be improved by establishing an off-line knowledge production process, without the excessive burden of the management of the central knowledge repository.

4 Conclusions – General Remarks

The concepts of the inner and outer context discussed above were empirically identified using a single case. Although it is dangerous to attempt any broad generalisations based on this kind of a study, we wish to close the paper by outlining some general remarks related to the contexts of KM based SPI.

First of all, we want to emphasise that to be able to recognise especially the key elements of the outer context and their change may be even more important than rushing into KM based SPI activities within some particular inner context. This is obvious from the case discussed above. The importance of the outer context is based

on the fact that the organisation's business goals affect directly the value expected to be created in the SPI and SWE processes – resulting in SPI business cases, as we have called them above.

Achieving the organisation's business goals requires not only that certain SPI and SWE processes can be conducted, but also that there is a relationship between the outer and inner contexts. The relationship may focus on specific resources, activities, or actors, and thereby affect the types of SPI solutions generated and the value expected to be gained. The case that we have discussed indicates that it is nothing but straightforward to set the goals and assess the value resulting from KM based SPI. Otherwise the case organisation would have selected the process optimisation approach much sooner, and ensured that it will result in some explicit business value.

Based on the analysis of the case, it is obvious that there is a grey area in between the inner and outer contexts that should be addressed in more details. What could be learnt from the case is that instead of pre-determined contextual models and fixed change paths, researchers should have some pre-understanding of the phenomenon and be equipped with a few conceptual tools. Subsequently, they would need to study the actual phenomenon during some period of time, to capture both its key elements and their change. Analysis of these elements and their change should result in a better understanding of the interplay of the contexts of KM based SPI. One of the key difficulties is that the knowledge involved in SWE and SPI is not easily identifiable and well-structured, and that the roles of various involved actors is still quite blurred.

Our view to the inner context was twofold, consisting of knowledge and software related parts. In the case organisation these contexts became intertwined, but could be made explicit through different processes and organisational responsibilities. Instead of, for example, management and sharing of tacit knowledge that have interested many KM researchers (cf. e.g. [2]), we took a view that emphasises explicit knowledge production in the knowledge context on the one hand, and systematic reuse of the knowledge in the software context on the other hand.

Although most of the knowledge in the case was produced as part of the SWE process in the software production organisation itself, there were severe problems in the acquisition and supply of the knowledge by the SWE organisation and in connection with the SWE process. The SPI organisation needed to take the role of a knowledge acquirer and broker with the help of an external expert organisation. Thereby KM became, to a large extent, an immediate just-in-time service for software development projects.

Instead of the original aim of the case organisation at setting up explicit KM processes and building the generic knowledge database – which did not succeed - a more focused approach was finally taken. Value is now sought from optimising the SWE process by design experience reuse in specific customer projects. In between there were attempts to materialise shorter-term gains by the maintenance and use of the knowledge repository by special knowledge brokers, but this appeared to be a difficult task. One of the key general remarks that we wish to point out to the members of the SPI community interested in making use of KM is thus, referring to Figure 3, to start from process optimisation rather than any of the other three areas of the outer context. Moreover, it is obvious that to remain in that corner is not a good idea, but process innovation is the SPI business case to achieve in the long run.

References

1. Wang, Y., King, G. Software Engineering Processes, Principles and Applications. CRC Press LLC, Boca Raton (2000)
2. Nonaka, I., Takeuchi, H. The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press. New York (1995)
3. Priest, Y. K. IBM's Reuse Programs: Knowledge Management and Software Reuse. The 5th ICSR, June 2 - 5 1998. Canada: IEEE (1998) 156-165
4. Davenport, T. H., Prusak, L. Working Knowledge: How Organizations Manage What They Know. Harvard College Business School Press. Boston (1998)
5. Carneiro, A. How Does Knowledge Management Influence Innovation and Competitiveness? In Journal of Knowledge Management, Vol. 4 (2). 87-98
6. Houdek, F., Kempter, H. Quality Patterns - An Approach to Packaging Software Engineering Experience. In Proceedings of the Symposium of Software Reusability (SSR'97) Software Engineering Notes (1997) 81-88
7. Messnarz, R., Stöckler, C., Velasco, G., O'Suilleabhain, G., Biro, M., Remszö, T. A Learning Organisation Approach for Process Improvement in the Service Sector. In Proceedings of the EuroSPI'99 Conference (1999) 12.13-12.30
8. Conradi, R., Dingsøyr, T. Software Experience Bases: A Consolidated Evaluation and Status Report, Lecture Notes in Computer Science, Vol. 1840. Springer-Verlag Berlin Heidelberg New York (2000) 391-406
9. Schneider, K. LIDs: A Light-Weight Approach to Experience Elicitation and Reuse. Lecture Notes in Computer Science, Vol. 1840. Springer-Verlag Berlin Heidelberg New York (2000) 407 - 424
10. Kucza, T., Komi-Sirviö, S., Utilising Knowledge Management in Software Process Improvement - The Creation of a Knowledge Management Process Model. To be published in Proceedings of International Conference of Concurrent Enterprising, Germany 27-29 June (2001)
11. Hammer, M. Champy, J. Re-Engineering the Corporation: A Manifesto for Business Revolution. New York. Harpercollins (1993)
12. Morris, D., Brandon, J. Re-Engineering Your Business. New York.. McGraw-Hill (1996)

Models and Success Factors of Process Change

Marion Lepasaar, Timo Varkoi, and Hannu Jaakkola

Pori School of Technology and Economics, Pohjoisranta 11,
PO Box 300, FIN-28101, Pori, Finland
{Marion.Lepasaar, Timo.Varkoi, Hannu.Jaakkola}
@pori.tut.fi

Abstract. Software process improvement is a popular way to increase the quality of software and the predictability of software engineering. In order to acquire effective processes, the software process improvement needs to be continuous. In this article we analyze the improvement models, which aim to introduce detailed actions for continuous process improvement and the key success factors of improvement based on various literary sources. As a result of the comparison of the two, we have pointed out issues, which, if considered in the improvement models, would help organizations in their improvement efforts.

1. Introduction

With software applications growing in their size and complexity, resulting in implementation difficulties and even project failures, software process improvement is becoming increasingly popular throughout the software industry.

Software process improvement (SPI) in small (up to 100 software engineering related positions) enterprises or business units requires special attention when applying models and standards which have usually been designed from the viewpoint of large organizations (over 250 employees). [12]

Software organizations, both software companies and software development units in large enterprises, involved in the SPI can apply a wide range of models that guide them through software process assessment, e.g. ISO/IEC 15504 [7], CMM [11], ISO 9000 [6]. As a result of assessment the organizations become aware of the weaknesses of their processes at the given time. A topic that has not been given the necessary amount of thought is how to improve those processes, especially in a continuous manner.

There have been various discussions [9, 5, 4] about what makes process improvement successful but not all these factors have been considered in the improvement models. Would it help organizations' software process improvement activities if the improvement models were more comprehensive and detailed?

In this article we aim to analyze and compare the key success factors listed in software process improvement related literature and the process improvement models. Although the key success factors vary based on the level of detail, the comparison results in features, which have not been considered in the improvement models. The two improvement models described in this article are the IDEAL model [3] and

ISO/IEC 15504 Part 7 [8]. The main reason for describing these models is their correspondence to the two widely used software process assessment models, CMM for Software and SPICE (ISO/IEC 15504), respectively.

The article is divided into three parts:

1. Analysis of the key success factors of software process improvement – analysis is based on factors described in four different literary sources;
2. Analysis of improvement models – description and comparison of the IDEAL model and SPICE Part 7;
3. Summary – the main findings of the comparison of key success factors and improvement models.

The article ends with short description of future research on the related subject.

2. Key Success Factors in Software Process Improvement

Throughout software process improvement (SPI) literature key success factors (KSF) have been listed, which would give maximum benefit to SPI activities. It is possible to divide the KSFs of SPI into two groups: 1. KSFs of successful SPI environment, and 2. KSFs of effective SPI activities in one company.

2.1. KSFs of Successful SPI Environment

In SPI related literature, the majority of sources imply success factors in correspondence with implementation in a one-organization environment. As a result of a comparison of regional initiatives, we have suggested in [10] that success factors in regional environment of SPI are more general and comprehensive. The four success factors found as a result of the analyses are described below.

First, public funding is important due to the lack of resources in small software organizations. Furthermore, it is often the public funding that enables the small software organizations to get involved in SPI initiatives. Later, when SPI activities have increased a company's productivity, external financial support is not necessarily needed. On the other hand, public funding can support the large-scale software process improvement programs effectively, like regional SPI programs.

Secondly, a learning environment is essential in order to support the continuous SPI activities. The learning environment as viewed in here means both the possibilities of getting knowledge and training about SPI methods and tools, as well as having a mentor to help in assessment and improvement activities. In order to establish a network of software companies involved in the SPI program the external consultant is vitally important. It is of especially great benefit when a regional SPI initiative can use the know-how and experience of similar activities.

Thirdly, the organizations that are ready to invest resources in SPI activities have already gained a certain maturity level since they have the necessary motivation, time, money and human resources to devote to it. The organizations ready for SPI activities have already understood and/or experienced the necessity of it.

Finally, there are different ways to view software process improvement. SPI can be aimed at achieving e.g. the ISO 9000 certificate as a result of an audit. At the same time efficient process improvement is a result of continuous work and not only a one-time effort. Whatever the process improvement model, the continuous SPI activities also prepare an organization to achieve any external quality certificate, in case the certificate is what the organization needs. Thus the fourth success factor is the establishment of continuity in the SPI work.

2.2. KSFs of Effective SPI Activities in One Company

There has been a lot of discussion in SPI literature about the success factors of SPI in one company. From Watts Humphrey to today's case-study analyses and feedback, we can see that the majority of the success factors are similar to each other. Following is the list of factors by Humphrey, Kautz, Grover and a feedback of one regional SPI initiative.

Humphrey has brought out several important issues worth considering for a successful process change in [5]. According to Humphrey, the basic principles of software process change are the following:

1. Major changes to the software process must start at the top – senior management leadership is needed for launching a change effort that also provides continuing resources and priority;
2. Ultimately, everyone must be involved – anyone who does not participate in improvement will miss the benefits of the change and might even inhibit progress;
3. Effective change requires a goal and knowledge of the current process – know where you are and where you want to be;
4. Change is continuous – software process improvement is not a one-time effort, it involves continual learning and growth;
5. Software process changes will not be retained without conscious effort and periodic reinforcement;
6. Software process improvement requires investment – investment like planning, dedicated people, management time and also capital investment.

Humphrey has also listed the key points of a successful software process improvement plan in [5]:

1. To improve the software process, someone must work on it;
2. Unplanned process improvement is wishful thinking;
3. Automation of a poorly defined process will produce poorly defined results;
4. Improvement should be made in small, tested steps;
5. Train, train, train. (Training is expensive but not nearly as expensive as not training. Training must be planned, funded, scheduled and required).

Kautz has listed the key success factors in his article [9], based on the experience gathered from SPI projects of three small software companies in Germany.

The factors as he listed and explained them are:

1. A flexible, tailored assessment and improvement approach;

2. A functioning network of small enterprises and their environment;
3. External technical help by mentors for change;
4. External financial support coupled to some conditions for performance.

In [4] Grover studied the evolution of the reengineering concept and its evolution toward the broader notion of process change management. As a result of the analysis of two studies that explore reengineering from a project implementation and an organizational perspective at two different points of time, he listed key factors that influence reengineering.

Grover points out the necessity of management's support to process change. Management could help in defining goals for the organization regarding change, providing strategic direction from the top and introducing well-implemented change management practices. Management's support is regarded as critical for success.

Grover also notes that continuous process management is being recognized as important, seems to be having positive impact on project outcomes, and tends to emphasize customer-oriented measures. Continuous process management practices are important in terms of their positive effect and are being implemented by an increasing number of organizations.

As a result of a feedback of a regional initiative of SPI called SataSPIN with over 10 participants (small software organizations of Finland), the key success factors in a prioritized list with the most important factor on the top, are the following:

1. SPI related training;
2. External guidance of the SPI work;
3. Company's commitment to SPI activities;
4. External support for SPI activities;
5. Management's support for SPI;
6. SPI environment support for a sufficiently long period of time (external mentoring);
7. Availability of information about SPI;
8. External financial support;
9. Availability of company's own resources;
10. Measurable targets set to SPI work.

It is easy to notice that the key success factors are almost the same in all four cases. Today it has become common knowledge that big changes need management support, resources, training, and external consultation. The most widely known fact is that you cannot start an improvement without knowing where you stand. Process assessment is needed in order to see the weaknesses of processes and to generate improvement ideas.

2.3. Summary of the KSFs

Following is the summarized list of KSFs. Some of the factors are mainly related to general change management issues, some to small organizations and some are SPI specific factors:

Table 1. Key success factors of software process improvement

General
Big changes start from the top and managerial support is required throughout process change
Finally everyone must be involved
Change requires investment and resources
Change related training is needed
Improvement should be conducted in small and tested steps
Process change needs to be continuous
Set measurable targets to improvement
Small organization specific
External financial support at the beginning
External guidance and mentoring
SPI specific
Assessment is needed before improvement can be conducted
Flexible and tailored approach to assessment and improvement

3. Software Process Improvement Models

Improvement models and guidelines have been created in order to help software organizations continuously improve their processes. These methods introduce steps that need to be taken for the entire SPI cycle. The best known improvement models in software engineering are IDEAL (originally created to support SPI related to Capability Maturity Model) and ISO/IEC 15504 Part 7 (providing guidelines for ISO/IEC 15504, also referred to as SPICE, based SPI).

Despite the differences in terms like model, improvement cycle or improvement guidelines, the aim of IDEAL and SPICE Part 7 are closely related to each other. IDEAL model aims to satisfy the organizations’ need for a specialized, systematic approach for managing technology adoption life cycle. SPICE Part 7 provides organizations with process improvement activities, which, if pursued in a consistent and disciplined series of steps, will permanently accumulate the benefits of SPI. Both SPICE Part 7 and IDEAL aim to describe continuous flow of improvement activities that would establish software process improvement cycle.

3.1. SPICE Part 7 – Software Process Improvement Guidelines/Cycle

SPICE Part 7 is primarily aimed at the management of an organization considering or undertaking a software process improvement program, possibly as a result of a process capability determination [8].

The guidance provides a framework for implementing improvements in a continuous manner. The concepts and principles are appropriate for the full range of different business needs, application domains and sizes of organization, thus fits to all types of software organizations to guide their improvement activities. [8]

The software process improvement cycle suggested in SPICE Part 7 includes the following steps:

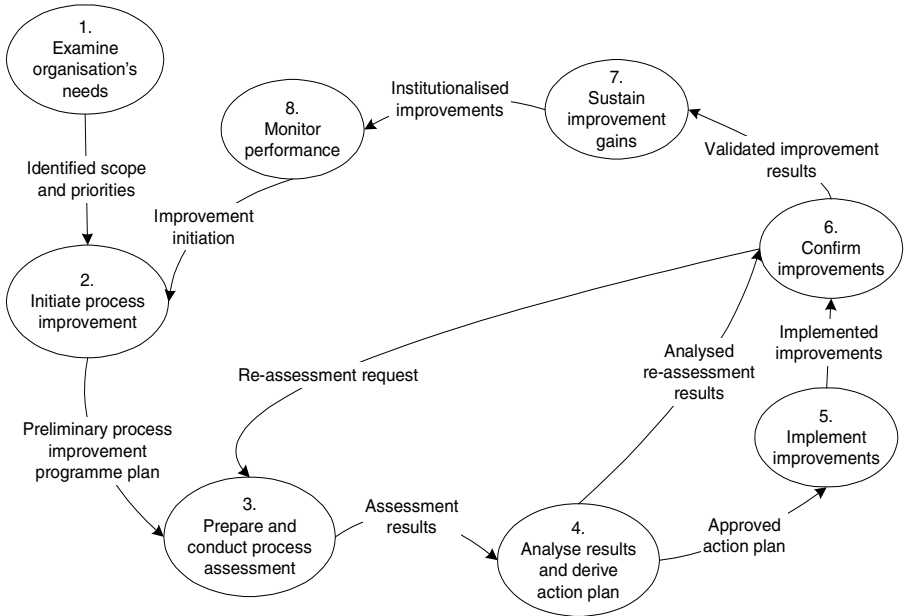


Fig. 1. Software Process Improvement cycle [8]

3.2. The IDEAL Model

The IDEAL model was originally conceived as a life-cycle model for software process improvement based upon the CMM for Software. The purpose of the IDEAL model, as it was originally conceived, was to describe in general terms the stages through which an organization involved in software process improvement would pass. An updated IDEAL model, revised by SEI for broader application of any improvement effort, provides a usable approach to continuous improvement by outlining the steps necessary to establish a successful improvement program. [3]

The IDEAL model aims to provide a disciplined engineering approach for improvement, focusing on managing the improvement program. The model also aims to establish the foundation for a long-term improvement strategy. [3]

The model consists of five phases, and each phase is further divided into steps. The initials of the phases form the name of the model:

- 1. I – Initiating phase
- 2. D – Diagnosing phase
- 3. E – Establishing phase
- 4. A – Acting phase
- 5. L – Learning phase

The phases of IDEAL are described within the steps below:

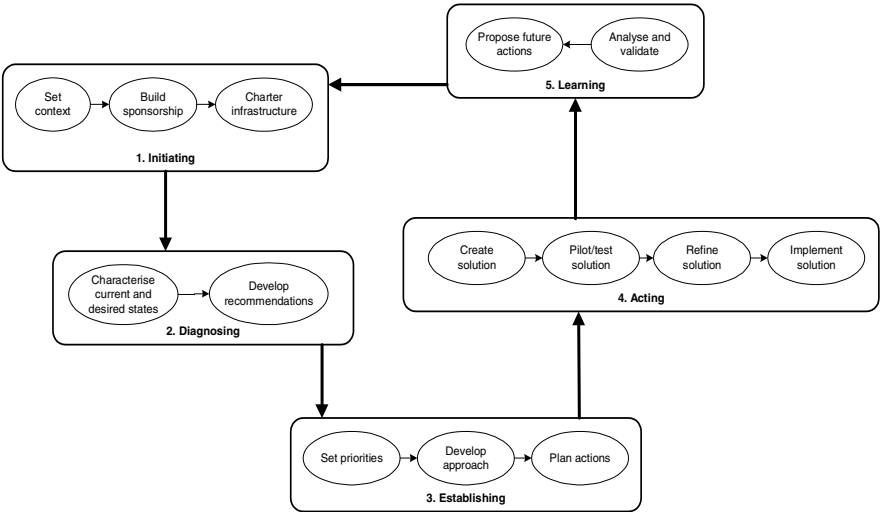


Fig. 2. The IDEAL model [3]

The figure describes the action of software process improvement, starting with the initiating phase, organization should follow the direction pointed by arrows. The last step of the final phase leads back to the initiating phase, thus forming a cycle of improvement.

3.3. Comparison of SPICE Part 7 and IDEAL Model

We have mapped the steps suggested by SPICE Part 7 against the steps of the IDEAL model. Despite the differences in the titles of the actions, we have compared the steps according to their context. The steps are compared based on the aims of the corresponding actions of the steps. The result of the comparison is described in the table where the parallel rows with gray background indicate largely similar actions of the two improvement models.

Table 2. Improvement steps of SPICE Part 7 and the IDEAL model.

SPICE Part 7	The IDEAL model
Examine organization's needs and business goals	Initiating phase
Formulation of mission statement or a long-term vision	Stimulus for change
Analysis of organization's business needs	Set context
Analysis of organization's current shared values	Build sponsorship
Organization's readiness to undertake process improvement program	Charter infrastructure
Data or quality costs	
Other external or internal stimuli	
Initiate process improvement	
Preliminary process improvement program plan	
Prepare and conduct process assessment	Diagnosing phase
Prepare assessment input	Characterize current and desired states
Conduct process assessment	Develop recommendations
Analyze assessment output and derive action plan	Establishing phase
Identify and prioritize improvement areas	Set priorities
Define specific improvement goals and set targets	Develop approach
Derive action plan	Plan action
Implement improvements	Acting phase
Select operational approach to implementation	
Prepare and agree the process improvement project plan	Create solution
Implement the process improvement action	Pilot/test solution
Monitor process improvement project	Refine solution
Confirm improvements	Implement solution
Improvement targets	Learning phase
Organizational culture	Analyze and validate
Re-evaluate risks	
Re-evaluate cost benefit	
Sustain improvement gains	
Monitor performance	
Monitoring performance of the software process	
Reviewing the process improvement program	Propose future actions

4. Summary

4.1. Key Success Factors and the Improvement Models

As a result of the detailed description of the KSFs and the models of software process improvement, we can notice some shortcomings of the improvement models.

Both the KSFs and the models convince us of the importance of senior management support to the SPI work. Humphrey has said that the big changes, like changes in processes, can only start from the top management. At the same time it remains unknown when and how senior management becomes aware of the necessity of SPI. It seems impossible that top management would support SPI while not being sure of its efficiency.

We can also see that in the listing of KSFs, training is one of the most important features. In improvement models training is not mentioned at all. At the same time, without training the management as well as the engineers would not accept nor be able to conduct the necessary process change. Without training the organization might also not apply the improvement model most suitable for them. In addition to that, an unprepared organization may relatively easily experience the failure of implementing process change.

Closely related to training is the factor of external guidance and mentoring of SPI. As a result of the scarcity of resources devoted to SPI, small software organizations require external SPI related consultation. Small organizations find external guidance necessary throughout the SPI activities. But the improvement models do not mention it.

Both the KSFs and the models repeatedly stress the importance of the improvement work being continuous. However, when following the SPICE Part 7 guidelines in improving the processes, we see no certain action that would establish the continuity in improvement activity. It would help the organizations if they decided upon a certain time period, after which some processes would be assessed and improved again, with the aim to establish continuity in SPI work. Unfortunately SPI is very often still viewed as one-time effort by the software organizations. At the same time, Zahran says in [13] that if process improvement is not actively pursued, the process could lose its effectiveness. A process without a proactive effort to improve it will decay.

4.2. Future Work

In future other models supporting process change and improvement will be studied, such as Balanced Scorecard [1] and European Foundation for Quality Management [2], to analyze the strengths and the shortcomings of these models. The further goal of this research is to suggest an improvement model that would best fit small software organizations on its appropriate level of detail. We will concentrate on describing, in greater detail, the complicated company-specific steps of software process

improvement - implementing the improvements, measuring the efficiency of SPI work, and establishing continuity in improvement work.

References

- [1] Balanced Scorecard Institute <http://www.balancedscorecard.org> (20.03.2001)
- [2] European Foundation for Quality Management <http://www.efqm.org> (18.02.2001)
- [3] Gremba J., Myers C. (1997) The IDEAL Model: A Practical Guide for Improvement. In Software Engineering Institute publication Bridge, nr.3.
- [4] Grover V. (1999) From Business Reengineering to Business Process Change Management: A Longitudinal Study of Trends and Practices. In IEEE Transactions on Engineering Management, Vol. 46, No. 1, pp. 36-46.
- [5] Humphrey W.S. (1990) Managing the Software Process. Series in Software Engineering, Software Engineering Institute, Addison-Wesley.
- [6] ISO 9000:2000: selection and use. <http://www.iso.ch/9000e/essai.htm> (22.03.2001)
- [7] ISO/IEC 15504-2:1998 Information Technology – Software process assessment – Part 2: A reference model for processes and process capability.
- [8] ISO/IEC 15504-7:1998 Information Technology – Software process assessment – Part 7: Guide for use in process improvement.
- [9] Kautz, K. (1998) Software Process Improvement in Very Small Enterprises: Does It Pay Off? In Software Process – Improvement and Practice, nr. 4 pp. 209-226.
- [10] Lepasaar M., Kalja A., Varkoi T., Jaakkola H. (2001) Key Success Factors of a Regional Software Process Improvement Programme. PICMET'01, Portland, Oregon.
- [11] Paulk M., Curtis B., Chrissis M. B. and Weber C. (1993) Capability Maturity Model for Software, Version 1.1. Technical Report CMU/SEI-93-TR-24, Software Engineering Institute.
- [12] Varkoi T. (2000) Software Process Improvement Priorities in Small Enterprises – thesis for the degree of Licentiate of Technology. Tampere University of Technology, Information Technology, Pori, Finland.
- [13] Zahran S. (1998) Software Process Improvement: Practical Guidelines for business success. Addison-Wesley.

Process Improvement in Turbulent Times – Is CMM Still an Answer ?

Karl Lebsanft

Siemens AG, Corporate Technology – Software & Engineering,
D-81739 München, Germany

`karl.lebsanft@mchp.siemens.de`

Abstract. In the volatile internet and e-business markets with a dominating factor 'speed' we have to face rapidly changing market and user requirements. Subsequently we have to give up development guidance by traditional – 'heavy weight' – processes to be able to react quickly enough. The big challenge is to find the right balance between clear orientation towards quality and reliability goals on the one hand and improving the ability to deliver to the market with higher frequency. Process improvement based on the Capability Maturity Model has demonstrated its benefit all over the world, but in these times there are voices claiming that CMM is no longer suited to the fast-paced, flexible, and innovative development required in the internet age. Experience at Siemens shows a changing world of IT business with consequences concerning process improvement. When properly used, CMM is still an excellent guide for successful software development –also in these 'agile times'.

Changing Conditions of Software Development

Software has become an integral part of today's business world. This is true not only for companies that sell software as a (stand alone) product or offer software-oriented services, but also for those companies that sell products and systems containing software as a significant part (mobile phones, process control systems, automobiles, etc.). Furthermore, software is used to run most business as such as well as to promote and sell products via the Internet. Due to these facts, software is increasingly synonymous with business itself.

Software development and moreover the whole software business currently are strongly influenced by the conditions and the needs of the new, growing, and changing internet and e-business markets. In these volatile and not yet settled markets, we face and have to react rapidly to changing market, customer, and user requirements – 'agility' has become the new buzzword denoting these crucial success factors.

Furthermore the dominating factor 'speed' in these environments also influence more and more the 'traditional' markets where software is of strategic importance, e.g. production automation, transportation, and health care. Also customers in these

‘traditional’ environments increasingly require agility from their suppliers – the ability to answer to changing or additional requirements, but still keeping their traditional requirements regarding quality and system performance.

Mastering the Software Process as a Precondition for Business Success

Over the past ten to fifteen years the ability to manage the software development process has been identified as crucial for success in the software business and much attention has been directed to improving the process capabilities of organizations which depend on high quality software. All over the world, the Capability Maturity Model (CMM) in these times has become a synonym for mastering and improving software processes. Process assessments and improvement programs based on the CMM have demonstrated their benefit in many major companies.

This is particularly true for Siemens where data collected in process assessments and re-assessments (around 250) over the past years, and more importantly the business results of the involved organizations, clearly show substantial improvement based on the guidance of the CMM. This is particularly true also for Siemens and its company wide Software Initiative. Data collected along with assessments and re-assessments over the past decade show substantial benefits of process improvement programs based on the CMM. Examples are:

- Reduction of development effort by 16-40%
- Effort for customizations reduced by up to factor 8
- Reduction in the number of errors by up to 62%
- Reduction in development and throughput time by up to 66%
- Significant increase in customer and employee satisfaction

Some discussions question whether the CMM as a guideline for process improvement is still the right way to deal with the requirements of today’s software business. There are voices claiming that CMM is only beneficial in ‘stable’ environments and not well-suited to the fast-paced, flexible, and innovative development that is required for success in the internet and e-business age. In the literature [7] there is a distinction made between ‘turbulent’ environments, where being able to react quickly to market needs and being innovative and flexible is of top priority, and ‘placid’ environments, where stability and having low defects is of top priority. It is concluded that the CMM is not applicable to the requirements of the more ‘turbulent’ high-tech developments. Furthermore, another claim that is sometimes heard is that there are successful and highly-visible innovative companies that don’t use the CMM and therefore it cannot be useful.

From our point of view, when looking closer at the criticisms it seems that they might be at least partially based on misinterpretations of the CMM. Sometimes it looks as if people put more attention on single parts or the wording of the CMM rather than to primarily try to understand and apply the ‘spirit’ of the model.

What is a Software Process ?

Our understanding of a ‘software process’ is quite simple. We want to organize the work that has to be done within software development in a way that ensures business success. Everyone should be precisely aware about the duties they are responsible for. Similarly, everyone should be clear about the contributions of their colleagues to the overall task and therefore about the results they can rely on. That’s all. How that clarity can be implemented when following a project through from collecting requirements until delivery to the customer differs a lot – depending on the markets, various customers’ behavior, different technology etc. – the appropriate solution depends on these and other factors.

What Guidance Does the CMM Provide with Respect to Changing Market Conditions ?

With the given process definition in mind and concentrating on the essentials, we still see a very fruitful application of the ‘old’ CMM in these ‘new’ times. In the following five examples – much more could be added – are discussed showing that the basic assumptions and requirements of the Capability Maturity Model are still valid, also for the so-called turbulent “Internet Markets”.

Orientation for People Participating in the Game – Shaping a Meaningful Process

The basic idea of process definition and improvement is to establish a common understanding of the necessary tasks and results of software development and the individual contribution of the participating roles and persons throughout the organization. This idea in the end holds for every kind of process and in particular for every kind of software production process independently from the side conditions encountered.

When we are confronted with a highly volatile market making it difficult to follow a clear and predefined way of performing development from the beginning to the end, there is a high probability to loose process rigor, even in cases where we would like to stick to commonly agreed working procedures. Loosing process rigor makes development less stable and predictable. Following the definition of a software process described above each persons certainty about their own duties and contributions from others decreases. When we regard process guidance positively, it is a natural conclusion to seek compensation for the lost process guidance.

Derived from our positive experience with development processes, we suggest to reinforce surrounding processes and people when an organization has to work under development conditions where sufficient process stability cannot be maintained due to market influences. A highly promising domain to be addressed here are the human / people issues. Applying e.g. the People CMM [12] to clearly define everyone’s role and to align personal goals with the goals of the organization may compensate at least partially for loosing orientation provided by the agreed software processes. Our initial experience with People CMM assessments and improvement projects reflects a large

and growing interest in this topic. By the way: orientation at the SW CMM and people issues seem to fit together. Along with CMM-based improvements, Siemens and also other companies have increased the employee satisfaction very visibly.

Focus Your Attention on Processes Relevant for the Most Critical Parts of Your Business

The software CMM demonstrates that a process is not a 'homogenous matter'. It is a collection of clearly defined and interrelated single activities, responsibilities, results, etc. Orientation towards business success therefore requires a focus on those process parts that strongly influence that success. For instance, when we have to deliver many versions and variants of a core product we should be strong in configuration management and configuration control. The business benefit of concentrating improvement effort to business relevant process aspects still holds in the Internet Age. Here we find that in emerging ('turbulent') markets it is even more important to have a close and efficient connection to the voices of the market and the customers than it is to produce software with ultimate quality attributes. When the novelty of product features is the basis of the customer decision rather than the non-functional properties of the product, we have to concentrate on getting the expectation from the customers right.

Along with the increasing importance of tracking and managing requirements of volatile markets we observe a generally increasing attention on the relevant organizational interfaces of software organizations to the market within Siemens: product management, marketing, sales. For this reason we consider it helpful to apply the basic CMM ideas – not the unchanged content of all the chapters – also in these parts of the organization.

At Siemens the business divisions product management and innovation management attract increasingly more attention. To be able to cope with product management issues and to guide the participants to straight forward and effective processes ensuring optimal alignment between market and development Corporate Technology has developed an assessment / improvement approach similar to its well known approach for development processes. First applications indicate a high potential benefit for the business division and confirm the value of this approach. To measure the ability of organizations to create innovative product and technological ideas and more importantly have them realized in products Siemens applies the "Innovation Cockpit" [3]. This shows the organization's innovation capability with respect to various metrics and provides guidance for optimizing "innovation processing".

CMM and eXtreme Programming

Another basic concept of the software CMM is to reduce the dependence of the organization on a few key people. Compensation for leaving key people never is easy but at least it should not cause a risk for the organization's success. The answer of the

CMM is to come up with a broadly agreed upon process that makes it easier to spread good practices across the whole workforce. Having this in mind it sometimes sounds curious when people discuss e.g. about substituting CMM based process work by 'Extreme Programming' (XP). From our point of view there is no contradiction between CMM and XP at all. For example, the extreme programming technique 'pair programming' which ensures that relevant knowledge regarding the software is not known only by a single person is fully in line with the CMM basics. And there are a lot more aspects of XP supporting CMM requirements (see also [5]).

Risk Management

Risk management is an issue being regarded in the Software CMM as part of Project Planning and Tracking. Risk management as an essential part of Project Management is crucial for avoiding negative surprises during project execution. And this for sure is true for every kind of business. One of the biggest risks that organizations face today is that they will be unable to respond quickly enough to unexpected challenges. Making good planning decisions early on and making sure that the development processes, software architecture, and organizational structure will be able to deal with the unexpected, is a genuine risk management activity. This ability to be 'agile' will pay off not only in the case where the organization must respond to an external situation but also can be used proactively to surprise the competition. Software engineering procedures that make this kind of agility possible are expected to be a key focus for the future.

There Are Projects and There Is an Organization Beyond the Projects

The Software CMM makes a clear distinction between project and organizational issues by assigning project related aspects primarily to the level 2 of the model and organizational aspects to the levels above. This distinction between projects and the surrounding organization again is something that has to be considered by organizations independently from the business they are in. What might change to some extent is the way of interaction between temporary projects and the lasting organization in which it is embedded. Our experience shows that projects in the agile age need full responsibility for their own issues to save time. There is no possibility (at least not always) for extensive discussions and long communication paths when a project-related decision has to be made.

On the other hand also the organization (and not only the projects) has to be more reactive when acting in the 'turbulent' environment. One strong requirement it has to satisfy is speeding up the whole business process. In rapidly changing markets a long term business strategy might not be possible at all. In accordance with the market the strategy has to be adopted much more frequently than in traditional businesses. And a higher pace in re-aligning the strategy requires a more frequent alignment of projects, products etc.

One of the major challenges for the organization in the turbulent environment is to determine the point in time when one has to switch from reacting to the market very flexibly and developing with only lightweight processes to the more stable and defined process management necessary for more established products (see also [6]). With a growing product history customers get familiar with specific features that no longer can be changed or left out again – more and more product content has to be maintained and quality requirements tend to increase. This typically requires both a maintainable and extensible software architecture as well as a maintainable and repeatable process.

What You Really Should Not Derive from CMM

The above presented examples show that the CMM basics still hold – even in a changing world of business. In the following on the other hand some typical misconceptions of the model are noted and discussed – again much more such examples are around.

You Have to Follow a Waterfall Model to Get a High CMM Rating

The CMM does not at all imply the usage of specific lifecycle models e.g. the well known waterfall model. It is true that it describes all the necessary activities of software development (design, implementation, ...) in a sequence but nowhere does it require this sequence to be followed in practice. We have applied CMM-based assessments to a good variety of different, often incremental, process models without any problems. The results show that the basic key process areas are of relevance independent of the lifecycle model. Note, however, that our experience shows that the reached maturity in the areas of configuration management, project management and requirements management have to be rather high to deal with incremental development successfully. Only stable organizations are able to implement incremental development under conditions requiring high quality and performance from the end product. Note also that the incremental approach is often not necessarily faster, but certainly more predictable and, due to its incremental nature, appears to be faster. Calling chaotic work ‘total flexible development’ only for a very short time will be enough to calm down the own management on one hand and the waiting customers on the other hand.

You Are Considered the More Mature the More Written Regulations You Have Available

In practice the CMM does not imply piles of process documentation. Procedures and methods have to be documented in adequate detail and clarity to be interpreted correctly particularly for new people on the project. Sufficiency of process documentation has to be rated according to the encountered level of common understanding among the organization’s workforce. If an organization is able to create

a common spirit of how to implement the project while only using a restricted volume of process documentation, this is still OK. We have found various organizations that successfully can manage and maintain their process with a very limited amount of paper. Note also that web-based solutions are increasingly successful in focussing the team members on the parts directly relevant for them. Often it is more important to convey the essence (and leave out some detail) in exchange for getting more people to read and live the described process.

People Are just a Cog in the Process Wheel – and Therefore Easily Replaceable

One of the most important aspects in fact not covered by CMM are people issues. But do not thereby conclude that the CMM could prevent you from dealing with your people well. Having ‘only’ a good process (definition) in isolation never will be enough. The process has to be filled with life – and only motivated people are able to achieve that. On the other hand we conclude from our experience with applying the CMM that increasing process maturity is a strong driver of people’s motivation – the higher the process maturity rating the higher the reported staff satisfaction in the organization. It seems to be more fun to work in an environment where everyone knows their responsibilities and those of their colleagues. And remember the paragraphs above: Having a sound understanding of the CMM also sharpens the view to the needs of people-related processes.

Conclusion

The current development in the software markets and businesses demands for increasing reactivity, flexibility and speed. When shaping the organization’s way of answering to these challenging requirements the ‘old’ Capability Maturity Model still offers valuable guidance. This is especially true when going back one step from the directly addressed process issues and thinking about the basics of the model. The model in fact does not cover all the aspects of a successful business, but it covers one essential part – the process – in a detailed and practical manner. A considerable share of the model’s assumptions and requirements can be transferred to other processes and aspects contributing to business success.

In Siemens we have good examples for very innovative and agile development organizations being in particular successful with CMM-oriented improvement. We also have convincing examples of projects on the other hand that were started under the label “creativity, flexibility, and innovation” with the interpretation: no defined process, little product documentation and little planning. All such projects, i.e. those that ignored the process aspects, failed if the number of involved developers was more than about seven and the expectation of the management was to get a product after a specified time (either total failure or mismatch of costs and schedule by more than 50%).

To summarize, we still consider the CMM to be an excellent basis for process improvement. It is definitely applicable in innovative as well as more traditional

environments. It does neither imply the waterfall process model nor it does require lots of process documentation. Its application for various organizations and products requires knowledgeable and experienced assessors and improvement coaches. It is not a step-by-step guide or a simple checklist that can be applied blindly. The Capability Maturity Model is not a cure-all (and it doesn't pretend to be one), but it is an excellent and unchallenged framework to structure the complex area of software development.

Literature

1. Kent Beck, "eXtreme Programming Explained" Addison-Wesley, 2000
2. Martin Fowler, "Put your Process on a Diet", Software Development, December 2000
3. Bodo Hasubek, "Whoever stands still falls behind: The Innovation Cockpit – Measurement and Control of the Innovation Capability", Software @ Siemens, March, 2001, <http://www.software-at-siemens.de/international/inhalt09.htm>
4. Steve McConnell, "Rapid Development: Taming Wild Software Schedules, Microsoft Press, 1996
5. Mark Paulk, "XP from a CMM Perspective", IEEE Dynabook on "eXtreme Programming: Pros and Cons – What Questions Remain?", 2000,
6. <http://www.computer.org/SEweb/Dynabook/PaulkCom.htm>
7. Judy Rothman, "Applying System Thinking to the Issues of Software Product Development", Systems Dynamics Conference, Cambridge, MA, 1996
8. Werner Mellis, "Software Quality Management in Turbulent Times – Are there Alternatives to Process Oriented Software Quality Management?", CONQUEST 1999, Nuremberg, Sept. 1999
9. Frances Paulisch and Axel Völker, "Best Practices for Software and Quality: The Techniques and their Business Benefit", The Second World Congress for Software Quality, Yokohama, Japan, 2000
10. Axel Völker, Karl Lebsanft, Frances Paulisch, "Agility – The Key to Success in Changing Markets and Technologies", ESEPG 2001, Amsterdam
11. Michael Cusumano, Sloan School of Management, MIT "Etablierte Firmen haben den Vertrauensvorschuss!", SiemensWelt 6/2001, pp. 44 – 45
12. Bill Curtis, William E. Hefley, Sally Miller, "People Capability Maturity Model", Software Engineering Institute, September 1995

Assessment of Maintenance Maturity in IT Departments of Public Entities: Two Case Studies

Macario Polo, Mario Piattini, Francisco Ruiz, and Mar Jiménez

Grupo Alarcos
Departamento de Informática
Escuela Superior de Informática
Universidad de Castilla-La Mancha
Ronda de Calatrava, 7
E13071-Ciudad Real (Spain)
Tel.: +34-926-295300 ext. 3706
Fax.: +34-926-295354
mpolo@inf-cr.uclm.es

Abstract. This paper presents the results of two capability maturity assessments done in two IT Departments belonging to public entities. We investigated their maturity in the Software Maintenance Process. The results obtained and the special characteristics of these organizations (produce, maintain and use its own software) invite us to do several reflections on the adequacy of maturity models to this kind of organizations.

1 Introduction

The assessment of software processes has become a common practice in many organizations. It provides a framework for evaluating the organization capabilities, allowing benchmarking and comparisons with respect to others. In this manner, the evaluated organization can know its weaknesses and strengths.

From the knowledge of its good and not so good practices, the evaluated organization is in conditions of putting into practice different types of measures (processes improvement) to reach higher levels of maturity. With this, organizations may estimate its technological level or know what kind of organizations can potentially become customers, since to reach and to stay at a certain level of maturity is an important added-value which may suppose a meaningful, enough guarantee for customers.

Therefore, to reach high maturity levels is important for software organizations; however, when the evaluated organization produces and consumes its own software, maybe some of the issues taken into account by assessment models are not so interesting for them as for organizations that develop software for a third-party. This is the case of IT Departments of public entities. Usually, these departments are repeatedly involved in similar projects, mainly related to the management of their relationships with citizens (taxes collection, vehicles enrolment, etc.) and of themselves (personnel management, documents management, etc.). Technical characteristics of these projects are also similar, since they use similar development environments and tools,

databases, etc. People in charge of these projects are also the same (mobility of public employees is quite less than those which work in private enterprises), being involved in all the stages of the life cycle of the software products: they analyze requirements, design the system, generate and test the code, put the system into operation and do its maintenance. Also the customer organization is the same, typically another group of employees in the same institution, although with different functions.

In the sense of [1], it is obvious that projects in these organizations have very high similarity. It is very possible they have all the characteristics to be at least at the Level 2 of the Capability Maturity Model (CMM).

In the other side, most of the work done by these organizations is maintenance (Singer reports that the 61% of the professional life of programmers is devoted to maintenance [4]).

For doing the assessment we used the IT Service Capability Maturity Model from the Vrije Universiteit [2]. As stated in [3], improvement models used for evaluating the Development process should be adapted to be used with other processes, as for example Maintenance. In fact, Maintenance has more characteristics of a service-oriented process more than product-oriented.

Assessments started with the Level 2 questionnaire, thinking in both cases that probably we should do later additional new assessments for Level 3. However, this last step was not done since results thrown by the Level 2 questionnaire placed both organizations in an uncertain situation between levels 1 and 2.

The IT Service CMM is specially adapted to Maintenance, as this process has characteristics enough to difference it from other processes, in such manner than Software Maintenance may be considered as a service more than as a product [3].

As it is well known, Level 2 at CMM is called "Repeatable", since organizations at this level repeat earlier successes with similar service levels for similar projects. As we have previously noted, the evaluated organizations have many characteristics that would do us to think about situating them directly at Level 2; however, the reality was very different.

In this paper we show the results of these assessments, as well as several reflections on them. The paper is organized as follows: in Section 2, we provide a brief description of the Level 2 questionnaire we have used. Section 3 describes the characteristics of both Data Centres and presents a summary of the results of the interviews with both organizations. An objective analysis of the results is shown in Section 4, using the interpretation guidelines found and inferred from [2]. In Section 5, we present some reflections and conclusions.

2 Maturity Questionnaire

The questionnaire used contains 65 questions related to the seven Key Process Areas (KPA) identified by Niessink and van Vliet for Level 2 [2]:

1. **Service Commitment Management.** Questions in this KPA check that the delivered services, the specified service levels and the customer's service needs are realistic and reviewed with the customer on a regular basis, adjusting the service level agreement when necessary.

2. **Service Delivery Planning.** The ten questions in this KPA checks that there is a plan for service delivery which is used as the basis for delivering the services.
3. **Service Tracking and Oversight.** This KPA checks the tracking of the service delivery (comparisons between specified and delivered services, corrective actions taken, etc.).
4. **Subcontract Management.** With this KPA, the selection of qualified subcontractors is checked.
5. **Configuration Management.** This KPA is used to test the politics in Configuration Management.
6. **Event Management.** This KPA is used to check how events (problem reports, etc.) are managed.
7. **Service Quality Assurance.** Through this KPA, the adequate management of quality procedures is evaluated.

The 65 questions of the questionnaire must be answered with:

- Yes always (YA)
- Not always (NA)
- Never
- Don't know

Also a "motivation" for the answer is solicited, in order to explain the circumstances which produce such answer.

3 Interviews

We maintained interviews with the responsible people of two IT Departments of two public entities: the Provincial Center of Informatics (CENPRI), belonging to the Provincial Council of Ciudad Real (Spain), and a second department of a Public organization that prefers remain anonymous.

The Provincial Council develops software for its own management, for the management of the 100 municipalities of the province and for managing its relationships with citizens. This includes taxes collection, properties seizures, public infrastructures, permissions for private works, salaries and personnel, etc. It also interfaces with several applications of banks, of the Ministry of Finance and of the region government.

The second Centre develops software for managing a big public institution, with several thousands of employees and also with many types of relationships with citizens and other institutions.

Both organizations use a tool for registering and tracking problem reports received from users. The anonymous organization uses a web tool for introducing them, whereas the Provincial Council receives requests by telephone or by a document.

Previously to performing the interviews we did an study of the distribution of modification requests received by these organizations per type of maintenance. In Table 1 the results for the Provincial Council is shown. This fact was one of the main reasons that invited us (both the authors and the responsible people of the IT Departments) to perform assessments of the maintenance process.

Table 1. Distribution of modification requests according to its type of maintenance.

Type of maintenance	% of MR
Corrective	13.30%
Perfective	62.21%
Adaptive	13.23%
Preventive	11.25%

Tables 2 to 8 show the seven Key Process Areas taken into account in the questionnaire. Their respective questions and the answers provided by both organizations are also shown. Only in some meaningful cases we have included a brief summary of the motivation. Furthermore, when the answers have been similar in both organizations, we have joint both responses into just a cell.

4 Analysis of Results

Key Process Areas for Level 2 are concerned with the establishment of the processes that enable the organization to repeat earlier successful services in similar situations. As it is seen in the previous tables, only the Provincial Council would be probably qualified as a “Level 2 organization”, since the other one has answered “Never” to near all questions.

Setting aside the evidence of the greater maturity of the provincial council, it surprises that satisfaction questionnaires passed to software users in both organizations produce similar results. It is important to take remember that these users constitute the customer organization of the IT Departments. Therefore, if customers are satisfied with the maintenance service provided, and maturity questionnaires do not highlight these results, maybe these ones should be adapted for considering the special characteristics of this kind of organizations.

In the following subsections, we propose some ideas that maybe should be taken into account in order to adapt this questionnaire (and maybe other questionnaires and assessment methods for other processes) to evaluate Level 2 maturity.

4.1 Experience of These Organizations

IT Departments of public entities produce always very similar software for the same people, to resolve similar problems; excepting new versions, development tools also remain across time.

All the KPA's of the questionnaire we have used contain one or more questions on the existence of documented procedures. As we see, only in the 5th KPA (Subcontract Management) both organizations coincide in the possession of documented procedures, and this is due to the obligation of contractual laws with public entities.

When an organization “always does the same”, with little personnel mobility, its experience in the development probably may substitute the existence of documented,

written procedures to do its work. We believe these questions should not be considered for these organizations at Level 2. We maintain only one exception in the aforementioned 5th KPA.

4.2 KPA “Service Commitment Management”

In our opinion, experience is also a factor with more importance for this KPA than the absolute tracking of service needs and commitments. Project managers, analyst and programmers know well characteristics of these repetitive programs: they quickly know and find where is the cause of an error, or what tables and modules must be changed to add a new functionality. The high similarity and the experience provides information enough to limit the controls in this KPA to event-driven assessments.

Certainly, emotional (and often physical) proximity of developers and users and the sharing of goals (“the institution must work”) facilitates the communication among both groups of people and the adequate resolution of problems.

4.3 KPA “Service Delivery Planning”

This KPA makes much insistence on the existence of documented procedures for a number of issues. This KPA is mainly related to the internal control of the IT Department. An interesting point to be taken into account for this KPA is that users always use the same version of a given software product; i.e.: if an error has been found in a program, the IT Department fixes it and updates all samples of that program. Moreover, in many cases the new version must be corrected or updated only on the server machine, since users employ terminals that are connected to a mainframe.

Therefore, some issues related to the service delivery plan may not be so important as in organizations that produce software for others.

This KPA also put special attention to the control of risks. However, similarity of the most influencing risk factors is so high, that both organizations coincide in the absence of risks in their projects. This comment (pronounced by our interlocutor at the Provincial Council) may seem exaggerated, but it is reasonably supported by the well-known problem domain, tools and people.

4.4 KPA “Service Tracking and Oversight”

This KPA contains questions for controlling some aspects related both to the relationship with the customer (mainly the questions: 1, 4 and 13) as to some internal issues of the service tracking and oversight.

We believe all issues in this KPA are important, even for organizations as those we are describing. In fact, the anonymous IT Department has, specially for perfective maintenance, a number of problems related to the need of overtime, which rarely appears in Provincial Council, that makes here a more rigid control.

4.5 KPA “Subcontract Management”

The importance of the right functioning of public entities for the daily life of citizens is so big, that we feel all relevant issues related to questions in this KPA must keep under the most absolute control.

As we see in Table 5, both organizations maintain their subcontractors under an adequate level of control. However, we think that the anonymous IT Department should make more emphasis in the planning and tracking of the service levels required to their subcontractors. This IT Department covenants service commitments with subcontractors, but only evaluates them when there are problems. This possibility should be avoided with a more rigid, periodical control of subcontractors.

4.6 KPA “Event Management”

Both organizations track adequately the events occurred and reported by users. Maybe experience could decrease the need of maintaining this control. However, event libraries and the related documentation constitute a very useful help when, for example, new personnel will be incorporated to the organization.

4.7 KPA “Service Quality Assurance”

Neither the Provincial Council nor the anonymous organization have adequate quality procedures. The reason of this is the great experience of both IT Departments and the high similarity of projects, tools and people. Spite this, and due precisely to the same exigency of public service mentioned in Section 4.5, public organizations should be much more strict with the Quality Assurance of their software.

In our case, this need is much more urgent in the anonymous organization: in this one, the people in charge of modifying a program is responsible of its testing, what is a unrecommended practice.

5 Conclusions

This paper has shown the results of the assessments realized in two IT Departments of two different public entities. These departments possess special characteristics that, in our opinion, should require specific maturity questionnaires and, maybe, specific adaptations of maturity models.

We have done some reflections on these adaptations. The main idea is that they should take into account the implicit “Repeatable” characteristics that these organizations have.

As future work, we are interested in using a more complete capability maturity model with a deeper taxonomy of maintenance activities as in the framework of the

CM³ model (Corrective Maintenance Maturity Model) developed by the Software Maintenance Laboratory in Sweden [5, 6].

References

1. Betiz, A. and Wieczorek (2000). *Applying Benchmarking to Learn from Best Practices*. Proc. of the International Conference on Product Focused Software Process Improvement (PROFES'2000). Lecture Notes in Computer Science, vol. 1840, 58-72.
2. Niessink, F. and van Vliet, H. (1999). *The Vrije Universiteit IT Service Capability Maturity Model. Technical Report IR-463*. Release L2-1.0. Faculty of Sciences, Division of Mathematics and Computer Science. Vrije Universiteit Amsterdam. Amsterdam, The Netherlands.
3. Niessink, F. and van Vliet, H. (2000). Software Maintenance from a Service Perspective. *Journal of Software Maintenance: Research and Practice*, 12(2), 103-120.
4. Singer, J. (1998). *Practices of Software Maintenance*. In Khoshgoftaar and Bennet (eds.), *Proceedings of the International Conference on Software Maintenance* (pp. 139-145). Los Alamitos, California: IEEE Computer Society.
5. Kajko-Mattson, M. (2000). *Corrective Maintenance Maturity Model*. Technical Report No. 00-010. Department of Computer and Systems Sciences (DSV). Stockholm University and Royal Institute of Technology.
6. Kajko-Mattson, M., Westblom, U., Forssander, S., Andersson, G., Medin, M., Ebarasi, S., Fahlgren, T., Johansson, S-E., Törnquist, S. and Holmgren, M. (2001). *Taxonomy of Problem Management Activities*. Proc. of the Fifth European Conference on Software Maintenance and Reengineering (CSMR' 2001). IEEE Computer Society, 1-10.

Acknowledgments

This work is part of the MPM project, carried out with Atos ODS Origin, S.A. and partially supported by Ministerio de Industria y Energía (CTR99-046).

Appendix

Table 2. KPA 1: "Service Commitment Management".

Question	Provincial Council	Anonymous org.
1. Are the IT service needs of the customer identified according to a documented procedure?	Yes, always (YA): a commission follows a docum. procedure	Never: there is no a documented procedure
2. Are the IT service needs of the customer documented?	NA (Not always): the planned ones are; the unplanned are not.	YA
3. Are the service commitments documented?	YA	Never
4. Are the service commitments evaluated with the customer on both a periodic and an event-driven basis?	YA	Never
5. Is the actual service delivery evaluated with the customer on both a periodic and an event-driven basis?	NA: only event-driven	

Table 3. KPA2: " Service Delivery Planning".

Question	Provincial Council	The other
1. Is the service delivery plan developed according to a documented procedure?	YA	Never: there is no a documented procedure
2. Is the service delivery plan documented?	YA	Never
3. Are the service delivery activities to be performed identified and planned according to a documented procedure?	YA	NA
4. Are software and hardware products that are needed to establish and maintain control of the service delivery identified?	YA	YA
5. Are estimates for the service delivery workload derived according to a documented procedure?	NA: there is no a written, documented procedure, but there is a "oral" plan known by everybody.	Never
6. Are estimates for the service delivery effort and costs derived according to a documented procedure?	Idem	Never
7. Is the service delivery schedule derived according to a documented procedure?	YA	NA
8. Are the risks associated with the cost, resource, schedule and technical aspects of the service identified, assessed, and documented?	Never: tools used, people, procedures, etc. are always the same, and risks are not evaluated. "There are no risks", he said.	Never
9. Are plans prepared for the service facilities and support tools?	YA	Never
10. Are service planning data recorded?	YA	NA

Table 4. KPA 3: "Service Tracking and Oversight".

Question	Provincial Council	The other
1. Is a documented service delivery plan used for tracking the service delivery activities and communicating status?	NA: status is verbally communicated; tracking always fulfils the same procedure, but is not documented.	Never
2. Is the service delivery plan revised according to a documented procedure?	NA: there is plan, but not documented	Never
3. Are approved changes to the service delivery plan communicated to the members of the service delivery group and other related groups?	YA	Never
4. Are actual service levels tracked against the specified service levels, and are corrective actions taken as necessary?	NA: there are revisions; they are not compared (there are no specified service levels)	
5. Is the service delivery workload tracked, and are corrective actions taken as necessary?	YA	Never
6. Are the service delivery activities' costs and effort tracked, and are corrective actions taken as necessary?	YA	Never
7. Are the service facilities tracked, and are corrective actions taken as necessary?	YA	Never
8. Is the service delivery schedule tracked, and are corrective actions taken as necessary?	NA	Never
9. Are the service delivery activities tracked, and are corrective actions taken as necessary?	NA	Never
10. Are the service delivery risks associated with cost, resource, schedule and technical aspects of the services tracked?	NA: only when new tools will be used or new people will be incorporated.	Never
11. Are actual measurement data and replanning data for the service recorded and made available?	YA	Never
12. Does the service delivery group conducts periodic internal reviews to track activity status, plans, actual service levels, and issues against the service delivery plan?	YA: one monthly	NA: there are reviews, but there is no a plan
13. Are formal reviews conducted with the customer to address the accomplishments and results of the services at selected moments according to a documented procedure?	NA: there are formal reviews, but there is no a documented procedure.	
14. Are formal reviews conducted internally to address the accomplishments and results of the services at selected moments according to a documented procedure?	NA: one monthly (question 12), but there is no a documented procedure.	NA: only when there are problems, but there is no a documented procedure

Table 5. KPA 4: "Subcontract Management".

Question	Prov. C.	The other
1. Is the service to be subcontracted specified and planned according to a documented procedure?	YA	
2. Is the subcontractor selected, based on an assessment of the subcontract bidders' ability to deliver the service, according to a documented procedure?	YA	NA: "ability is a factor, but there are much more"
3. Is the contractual agreement between the prime contractor and the subcontractor used as the basis for managing the subcontract?	YA	
4. Is the documented subcontractor's service delivery plan reviewed and approved by the prime contractor?	YA	
5. Is the documented and approved subcontractor's service delivery plan used for tracking the service activities and for communicating status?	YA	
6. Are changes to the subcontractor's service commitments, service delivery plan, and other commitments resolved according to a documented procedure?	YA	Never
7. Are subcontract service commitments evaluated with the subcontractor on both a periodic and an event-driven basis?	YA	NA: not periodically, only event-driven
8. Is actual service delivery of the subcontracted services evaluated with the subcontractor on both a periodic and an event-driven basis?	YA	NA: not periodically, only event-driven
9. Are formal reviews conducted with the subcontractor to address the accomplishments and results of the services at selected moments according to a documented procedure?	YA	Never
10. Does the prime contractor's service quality assurance group monitor the subcontractor's service quality assurance activities according to a documented procedure?	YA	Never
11. Does the prime contractor's configuration management group monitor the subcontractor's configuration management activities according to a documented procedure?	YA	Never
12. Does the prime contractor's event management group monitor the subcontractor's event management activities according to a documented procedure?	YA	

Table 6. KPA 5: "Configuration Management".

Question	Provincial Council	The other
1. Is a configuration management plan prepared for each service according to a documented procedure?	NA: only for the most important services	Never
2. Is a documented and approved configuration management plan used as the basis for performing the configuration management activities?	YA	Never
3. Is a configuration management library system established as a repository for the configuration base-lines?	YA	Never
4. Are the products to be placed under configuration management identified?	NA: only some products	Never
5. Are action items for all configuration items/units initiated, recorded, reviewed, approved, and tracked to closure according to a documented procedure?	NA, because there is no a documented procedure	Never
6. Are changes to configuration baselines controlled according to a documented procedure?	NA, because there is no a documented procedure	Never
7. Are (software) products from the configuration baseline created and released according to a documented procedure?	NA, because there is no a documented procedure	
8. Is the status of configuration items/units recorded according to a documented procedure?	NA, because there is no a documented procedure	
9. Are standard reports documenting the configuration management activities and the contents of the configuration baselines developed and made available to affected groups and individuals?	NA: there are no standards.	Never
10. Are configuration baseline audits conducted according to a documented procedure?	NA, because there is no a documented procedure	Never

Table 7. KPA 6: "Event Management ".

Question	Provincial Council	The other
1. Is an event management plan prepared for each service according to a documented procedure?	YA	Never: there is no plan
2. Is the documented and approved event management plan used as the basis for performing the event management activities?	YA	Never
3. Is an event management library system established as a repository for the event records?	YA (there is a tool)	
4. Are events identified, recorded, analyzed, reviewed, and resolved according to a documented procedure?	NA, there is no a docum. procedure	YA
5. Are affected groups and individuals informed of the status of events on both a periodic and event-driven basis?	YA	
6. Are standard reports documenting the event management activities and the contents of the event repository developed and made available to affected groups and individuals?	YA	
7. Are event repository audits conducted according to a documented procedure?	NA, because there is no a documented procedure	

Table 8. KPA 7: "Service Quality Assurance".

Question	Provincial Council	The other
1. Is a SQA plan prepared for the service delivery according to a documented procedure?	Never: there is no SQA plan neither documented procedure	
2. Are the SQA group's activities performed in accordance with the SQA plan?	NA, because there is no a documented procedure	Never
3. Does the SQA group participate in the preparation and review of the service commitments and service delivery planning, standards and procedures?	NA: there is no a specific SQA group, but people participate	Never
4. Does the SQA group audit the service delivery activities to verify compliance?	NA: there is no a specific SQA group, but there are audits	Never
5. Does the SQA group periodically report the results of its activities to the service delivery group(s)?	NA: reports are done, but not by the SQA group because there is not	Never
6. Are deviations, identified in the service activities and delivered service, documented and handled according to a documented procedure?	Never, since there is no SQA plan	
7. Does the SQA group conduct periodic reviews of its activities and findings with the customer's SQA personnel, as appropriate?	Never: there are non-periodic reviews; there is no SQA group	

Evaluating a Usability Capability Assessment

Netta Iivari and Timo Jokela

University of Oulu, Department of Information Processing Science
P.O. Box 3000, 90014 Oulu, Finland
{Netta.Iivari, Timo.Jokela}@oulu.fi

Abstract. Improving the position of user-centered design (UCD) in development organizations is a challenge. One approach to start improvement is to carry out usability capability assessments (UCA). We experimented a UCA in a software development company, and evaluated its usefulness. The evaluation consisted of (1) an analysis of the context of use of UCA; (2) definition of the evaluation criteria; and (3) performing the evaluation by using questionnaires. The results indicate that UCA should not only provide information about the position of UCD, but also motivate the personnel to experiment with it in their work, and provide training for them. In the UCA conducted, we succeeded in identifying the strengths and the weaknesses in the development process. The staff became motivated and had a positive attitude towards UCD. UCD was made highly visible in the organization. However, we identified also targets for improvement. The terminology related to UCD needs to be explained very carefully and a common terminology with the personnel needs to be developed. In addition, the UCA should focus on the needs of usability specialists and provide support especially for their work. UCA should also be done in an efficient way.

1 Introduction

Usability capability assessments (UCA) are conducted in order to define the usability capability of software development organizations. Usability capability is a characteristic of a software development organization that determines its ability to constantly develop products with high-level usability [13]. UCA has its basis in the software development world. *Software process assessments* have been introduced for analyzing the current organizational status of the software development. By performing process assessments one can identify the strengths and weaknesses of an organization and use the information to focus improvement actions. Analogously UCA aims at identifying focuses for improvement actions. Through a UCA one can identify major strengths and weaknesses in the organization in connection to user-centered design (UCD).

However, the position of UCD in the software development organizations is often ineffective. There has been intensive work going on over ten years to develop methods

and tools for supporting the implementation of UCD. Still there exist few organizations that consistently and systematically apply UCD in their development. The improvement of the position of UCD is widely recognized as a challenge [1], [13], [17].

This paper describes a UCA conducted in a large Finnish software development company, which develops software intensive products for the mobile networks for international markets. This was a third assessment conducted within a KESSU-research project, which aims at developing methodologies for introducing and implementing UCD in software development organizations. The UCA was performed for a User-Interface Team (20 people) in the company.

There does not exist many reports about evaluating the success of UCA's. The existing literature is focused on presenting the structure of the assessment models and the methods for performing the assessment. Our aim is to develop the assessment method as well as to improve the usability capability of the organizations. Previous assessments conducted suggest that there exist problems in the traditional process assessment method when applied in UCA. Therefore, feedback from the organization assessed and evaluation of the assessment method were in a central position.

In the next section we describe the assessment approach used. The approach has been modified after the previous assessments conducted. Then we describe how we planned to evaluate the assessment. The evaluation was conducted by interviewing the personnel of the organization, and based on that, the user requirements for the UCA were defined. In the following section, we report how we carried out the assessment and reported the results. Thereafter we report the procedure and the results of the evaluation. In the final section, we summarize our results, examine the limitations of the results, give recommendations to practitioners, and propose paths for further work.

2 The Assessment Approach

There is a relatively small community working in the area of UCD process assessment models. However, a number of different models have already been introduced [13]. Recent work on the models has been carried out also in standardization bodies. A technical report ISO/TR 18529: Human-centered Lifecycle Process Descriptions [11] was published in June 2000. A pre-version of the technical report, Usability Maturity Model (UMM): Processes [4] was developed in the European INUSE project and further refined in the TRUMP project. QIU – Quality In Use – [2], [3] assessment model is also based on the models presented above, but is somewhat wider in scope.

These models have their basis in two recognized sources: ISO/TR 15504 [7], [8], [9], [10] and ISO/IS 13407 [12]. The structure of the models is based on the format of the ISO 15504. ISO 13407 forms the basis of the content of these models. The UMM and ISO/TR 18529 models define a new (sixth) process category in addition to the

ones defined in software process assessment model ISO/TR 15504: *Human-centered Design, HCD*¹ HCD contains seven processes of UCD, as shown in Table 1.

Table 1. User-centered design processes

HCD.1 Ensure HCD content in system strategy
HCD.2 Plan the human-centered design process
HCD.3 Specify the user and organizational requirements
HCD.4 Understand and specify the context of use
HCD.5 Produce design solutions approach.
HCD.6 Evaluate design against requirements
HCD.7 Facilitate the human-system implementation

Each process is defined with a *purpose statement*. In addition, there are identified a set of *base practices* for each process. Each of the processes is assessed independently by concentrating on the base practices of the process. The capability of the processes is determined by using the scale defined in ISO 15504. Typically the results of an assessment are reported as capability levels of the processes. In all, the assessments based on these models are conducted according to the method defined in [8], [9].

We decided to carry out this assessment somewhat differently from the earlier ones, which were mainly carried out by following the 'standard way' of performing a process assessment; by examining the base practices of the processes, scoring them and examining the capability levels. We have gathered feedback from the assessment team and personnel of the target organization in the previous assessments [5], [6], [14]. The material highlights certain problematic issues in the assessment method used:

1. The base practices and the capability levels of the assessment model were difficult to interpret. As a result, it was difficult to interpret the results against the model.
2. A lot of the time in the interviews was put on issues that could have been found from the project documentation.
3. The target organization expected to have more concrete results.
4. The personnel of the organizations had some difficulties in understanding the results and the terminology used

This time our strategy was to experiment a new approach in situations where we earlier had found problems. One driving force was to have a clear interpretation of the model used. The lead assessor prepared an interpretation of each process to-be assessed beforehand. ISO 13407 [12] was used as a reference, but also [2], [3], [4] and

¹ We use human-centered and user-centered as synonyms.

[11] offered guidance. The interpretations led step by step towards a new definition of the processes. This - and the insights gained through the evaluation procedure described in the following section - led step by step towards a new assessment approach.

3 The Evaluation Framework

The basic purpose of an assessment is the identification of weaknesses and strengths of the UCD processes in a development organization. Thus, one viewpoint to evaluate the success of an assessment is to examine whether we really can reveal the relevant strengths and weaknesses, which should give guidance for the improvement actions of the organization. However, we understand that the success of an assessment should be evaluated from other viewpoints as well. 'Usability capability assessment method' is an artifact. Constructive research is about building artifacts for specific purposes, but also about evaluating how well they perform. [15]

Papers concerned with the assessment models and methods seldom approach them according to the principles of constructive research. One research activity of the constructive research is evaluation. According to the paper by March and Smith on research methods, "evaluation of artifacts considers the efficiency and effectiveness of it, and its impacts on the environment and its users". In other words, the evaluation of artifacts includes the (1) definition of "specific tasks", (2) development of metrics, and (3) comparison of the performance. [15]

We decided to perform the evaluation of the UCA a user-centered way. ISO 13407 [12] defines usability to be 'the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use'. In the context of assessments, we similarly emphasize the importance of identifying the users, context of use, and the goals, which the users wish to achieve. Those need to be achieved with effectiveness, efficiency and satisfaction.

We applied the idea of *context of use analysis*, which is an essential activity in user-centered design [12]. In relation to the '*specific tasks*', the *tasks* alone are not adequate to describe the use of artifacts. We also need to consider the *characteristics of users* who perform the tasks, and the *environment* in which the tasks are carried out [12]. We conducted the evaluation of the UCA method a user-centered way by

- (1) performing a context of use analysis of the assessment;
- (2) defining the evaluation criteria and metrics for the assessment. They were derived from the material collected during the context of use analysis; and
- (3) performing the evaluation by using questionnaires.

4 Carrying out the Assessment

There were altogether five members in our assessment team. The lead assessor was a trained process assessor. Two members of the assessment team were usability experts of the company. The assessment consisted of the following steps:

1. Examination of the project documentation: a document assessment was performed.
2. Interpretation of the model and interviewing during a period of two weeks
3. Presentation of the results to usability specialists and sponsor of the assessment.

The results were reviewed and the presentation refined

4. Presentation of the results to a larger audience
5. Preparation of the assessment report

The focus of the assessment was a typical project of the User-Interface Team. Based on the experiences of previous assessments, it was decided to examine the relevant documents first. This was done together with the usability specialists.

The part of the assessment process, which was visible to the whole personnel of the organization, started with an opening briefing in which the assessment team and the assessment approach were presented. Then we interviewed the personnel of the company (5 interviews). In each interview, there were one or two interviewees. The interviewees were persons who had worked in the selected project or were usability specialists of the organization. We discussed the selected project from the viewpoint of certain UCD processes.

A characteristic feature of the assessment was that before each interview, the lead assessor presented a precise interpretation of the model to the assessment team. Four main engineering processes were identified: *context of use process*, *user requirements process*, *user task design process*, and *usability evaluation process*². In relation to all processes, required outcomes and integration to other processes were defined.

We decided to conduct the assessment by focusing on the outcomes instead of base practices. First we examined the existence of a process. This was done through examining the *extent of the outcomes* of the process. The more extensively a process produced the outcomes, the higher performance score it got. If the process had a reasonable extent of outcomes, then we examined the *integration of the outcomes* with other processes. The more extensively the outcomes were communicated and incorporated into other processes, the higher rating was given to the integration. A third aspect was to examine the *quality of the outcomes*: how they were derived, e.g. were recognized user-centered design methods and techniques used while generating the outputs.

This assessment approach led to a different way of presenting the results. The goal of the presentation was on one hand to be communicative, on the other hand to have a positive tone (even if the results were anticipated not to be very good). One guideline was to present the results graphically. We rated the capability of the processes in the three dimensions with a scale: 0 – 5 (0 = not at all, 5 = fully). The results were presented both quantitatively and qualitatively.

² We have taken the names from the QIU model [2], [3].

During the assessment a modified assessment approach evolved. The presentation of the results, the interpretations of the processes and the concentration on the outcomes - not on base practices – developed the assessment approach towards a new direction. At the same time, the approach was evaluated. Through the evaluation process the context of use of the assessments was identified, and the goals the users wish to achieve through assessments were defined. Finally, the new approach was evaluated against the user requirements obtained through the evaluation process. Next section presents this process in a detail.

5 Evaluating the Assessment

5.1 Context of Use Analysis

In the beginning of the analysis of the context of use we identified the relevant user groups of the assessment (from the point of view of the target organization):

- Line Managers
- Project Managers
- Designers
- Usability specialists

The analysis of the context of use describes the characteristics of the users, the tasks the users are to perform and the environment in which users are to use the UCA method. We gathered the information by interviewing the personnel, including persons from all the user groups identified above. Altogether we interviewed 5 persons from the User-Interface Team. In each interview we addressed the following issues:

- Background: education, training and knowledge of user-centred design and process assessments
- Job descriptions and defined tasks
- Characteristics of the organization (positive and negative aspects)

We took notes in the interviews. Afterwards a comprehensive description of the context of use was produced. We organized a brainstorming session in which we categorized the material achieved using post-it notes on the wall, into the four user groups. Each category was further divided into three sections (characters, tasks, environment), as shown in Table 2. In the end of the session the table contained several notes related to each user group divided further to one of its sections: characters, tasks or environment.

Table 2. The analysis of the context of use

	Line Managers	Project Managers	Designers	Usability Specialists
Characters				
Tasks				
Environment				

5.2 Definition of the Evaluation Criteria

We defined requirements for the assessment by analyzing the material produced during the analysis of the context of use, again in a brainstorming session using the post-it notes on the wall. Each user requirement had to be derived from the description of the context of use. The session produced an abundant set of user requirements in relation to all user groups. The requirements had also to be related to one of the sections: to the characters, tasks or environment.

Afterwards the requirements were prioritized. The relevance and feasibility of the requirements were defined. We produced a list of user requirements containing 11 requirements (presented in the section 5.3 Performing the Evaluation). The requirements had to be validated by the users – the personnel of the organization to be assessed. We organized a meeting in which the persons interviewed were present. They commented on the requirements and some refinements were made.

We evaluated the assessment by using questionnaires. The questionnaires were constructed the way that they were able to address the user requirements presented above. The questionnaires consisted of both the open-ended and structured questions. The structured questions contained an ordinal scale (1 - 5, 1 = poor, 5 = good) for ranking certain objects with respect to certain characteristic (usefulness, understandability etc.). Open-ended questions addressed the same topics. They provided useful information about the motives for responding in a certain way. They provide useful input especially for the improvement of the UCA method.

5.3 Performing the Evaluation

The evaluation was performed by using questionnaires delivered to the audience each occasion the assessment team was in contact with the personnel of the company:

- In the opening briefing.
- After each interview
- In the presentation of the results

In the questionnaires delivered at the opening briefing we examined e.g. how understandable and interesting the presentation was, how clearly the purpose and the

procedure of the assessment were presented and whether the presentation offered new information or ideas for the audience. In the questionnaires delivered after each interview we examined e.g. whether the interview handled meaningful issues and whether the interview offered useful information related to the interviewee's work. Finally, in the questionnaire delivered to the audience at the presentation of the results the respondents evaluated the results, the presentation and also the whole assessment process and its usefulness, efficiency etc.

Altogether we received back 27 responses during the sessions mentioned above. Results are presented in Figure 1. More detailed descriptions of the responses are reported in relation to each user requirement.

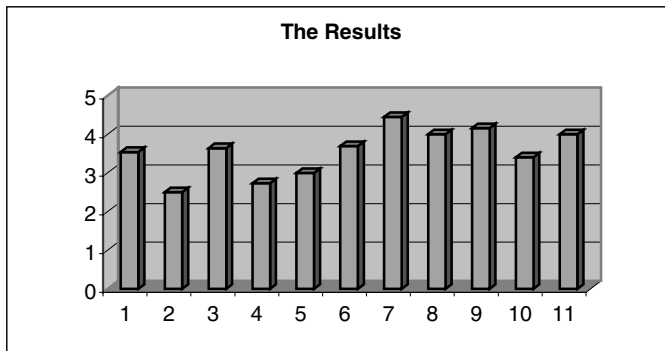


Fig. 1. Results from the questionnaires

1. The assessment must offer new information about the state of user-centered design in the organization (also for usability specialists). The assessment must provide a useful basis for improvement actions
 - The assessment offered reliable and meaningful information about the state of user-centered design in the organization
 - Usability specialists however claimed that the assessment did not offer that much *new information* – they had anticipated the results
 - The outlining of the results nevertheless offered new and useful basis for improvement actions
2. The assessment must identify targets for training
 - The assessment did not succeed very well in identifying targets for training. More detailed information would have been needed
3. The assessment must be useful experience for all involved
 - Variation in the responses was huge. However, issues concerning user-centred design were mentioned overall to be very meaningful and important. It was also very useful to think back and reflect on own ways of working. The interviews were thanked for being eyes opening, educational and useful experiences
4. The assessment must offer useful training
 - The interviews succeeded in having an educational function, the opening briefing and presentation of the results sessions not as much

- The presentation in the opening briefing did not offer much new information or ideas because almost all of the audience were already familiar with the subject
- Some interviewees felt they gained many new and useful ideas while others did not. Several issues were mentioned, which were learned during the interviews. Those were very concrete issues related to user-centred design or to the shortcomings in the old ways of working
- 5. The assessment must support the process improvement actions of the organization
 - The respondents claimed that the results should be even more concrete. The results mainly supported the usability specialists' own views about the state of user-centered design in the organization.
- 6. The assessment must produce input for the user-interface development strategy
 - The assessment succeeded in offering input for the strategic planning by reliably describing the current state
- 7. The assessment must motivate personnel to gain training in user-centered design and to experiment with it in their own work
 - The respondents were very interested in gaining training in user-centered design. They were even more interested in to experiment with it in their work
- 8. The outputs of the assessment must be presented in an understandable and interesting format
 - The outputs of the assessment were presented in a very understandable and interesting format
 - Especially the presentation in the opening briefing was very understandable. The purpose and the procedure of the assessment were presented very clearly.
 - The graphical presentation of the results gave quickly an overview of the results. The presentation was short enough and very clear. Maybe even more detailed explanations of the results could however be included
- 9. The assessment must emphasize the importance of the activities and principles of user-centered design
 - The assessment succeeded in emphasizing and highlighting the importance of the activities and principles of user-centered design very well.
 - All respondents – not only usability specialists – evaluated user-centered design to be very important. According to the responses user-centered design should have very visible and effective role in the development projects
- 10. The assessment must support the work and role of usability specialists in the organization and in the projects
 - According to the usability specialist the assessment did not considerably support their role in the organization or in the projects. However, the assessment made their job highly visible. The rest of the personnel also defined user-centered design processes to be very important, which can be seen as a support for the work of the usability specialists
- 11. The assessment must be conducted in a way that does not unnecessarily burden the organization being assessed
 - The assessment was thanked as being conducted very effectively and efficiently. Only the usability specialists had to put a lot of time and effort into the assessment

6 Discussion

We experimented with a UCA in a large Finnish software development company. The assessment was conducted differently from the previous assessments within the KESSU-research project. The previous assessments were conducted mainly by following the standard procedure of software process assessments. The need for improvement of the assessment method had evolved during these assessments. This time we refined the UCA method, and tried to improve the method a user-centered way, to meet the needs of the target organization.

The evaluation of the assessment was in a central position for several reasons. One reason was to improve the refined UCA method. From the point of view of constructive research UCA method is an artifact. Constructive research is about building artifacts for specific purposes, and evaluating how well they perform. [15] The evaluation was done in order to reveal how well the new artifact performs. The evaluation was conducted a user-centered way. The evaluation consisted of (1) the analysis of the context of use; (2) definition of the user requirements; and (3) conducting the evaluation by using questionnaires.

Through the evaluation process, we were able to identify users of the assessment, their characteristics, their tasks related to the assessment, the goals they wanted to achieve, and the characteristics of the environment the assessment was conducted in. The analysis of the context of use provided extremely useful information for the improvement of the assessment approach.

Altogether, the evaluation revealed that the UCA method succeeded well in meeting the user requirements. We identified the strengths and weaknesses in the development processes. The results offered a reliable basis for improvement actions. The results offered also a new and useful outlining of the UCD related activities, which provided additional value to the usability specialists, who otherwise anticipated the results. The results were presented graphically and within several dimensions (extent, integration and quality of the outcomes). The presentation was thanked for being very understandable and interesting.

It also turned out that the staff became motivated for the improvement actions, and estimated UCD related issues and activities to be very important in the organization. The personnel were very motivated for gaining training in UCD and experimenting with it in their own work. One benefit of the assessment was also the visibility that the usability specialists' work gained. The whole assessment process – not only the results – can be seen as motivators for the improvement actions in the organization.

However, we identified also targets for improvement. The terminology related to user-centered design needs to be explained very carefully. A common terminology with the personnel of the organization needs to be developed. Certain buzzwords exist in software development organizations. Without clarifying them the assessors have difficulties in interpreting the findings and presenting the results. The importance of the personnel of the organization for attending the opening briefing and the presentation of the results needs to be emphasized. Only in this way can the educational and motivating aspects of the assessment be utilized. Especially the interviews provided good opportunities to educate the personnel in UCD.

The results of the evaluation confirm our observations from the previous assessments [5], [6], [14]. The understandability of the terminology related to UCD and the assessment is very important. Overlooking it can seriously hinder the success of the assessment. In addition, UCA should not be seen as consisting of giving capability ratings only. Instead, UCA needs to be seen as an occasion in which the personnel of the organization can be educated and motivated for the improvement of UCD. The results of a UCA are usually not very good. The position of the UCD is often nonexistent or ineffective in the organizations. Due to this, the assessments solely aiming at rating the process performance do not fit well in this context. Furthermore, the outcome based assessment model provided a clear and unambiguous basis for the assessment. Base practice driven assessment left too much room for the interpretations.

However, the evaluation process succeeded also in providing new insights related to the UCA. The usability specialists have not been as clearly in the focus in the previous assessments. The interviews succeeded in showing that the main users of the assessment were the usability specialist, whose work the assessment should particularly support. The assessment provides value to the organization not only by educating the whole personnel, but especially by supporting and educating the usability specialists. Another aspect not realized in the previous assessment was the emphasis on the efficiency of the assessment. The assessment should not demand a lot of resources. While acknowledging this requirement, we reduced the amount of interviewees and lightened the whole assessment procedure. Still we think, that the assessment succeeded in providing as good results as the previous assessments.

The assessment was recently performed, and thereby we do not yet know how the organizational improvement actions will truly take off. The feedback was positive, but the true validity can be evaluated only in the future. On the other hand, the success of improvement actions will depend on many other things – not only on the assessment and its results. Our main recommendation based on this experiment is that carrying out usability capability assessments is useful and gives a good basis for improvement actions in UCD. However, the assessment approach needs tailoring. The traditional process assessment approach used in software development is probably not always applicable in assessing UCD. One example of such tailoring is presented in this paper.

References

1. Bloomer, S. and S. Wolf. Successful Strategies for Selling Usability into Organizations. in Companion Proceedings of CHI '99: Human Factors in Computing Systems. 1999. Pittsburgh, USA: ACM Press.
2. Earchy, J. Quality In Use processes and their integration - Part 2 Assessment Model, Lloyd's Register, London, 2000a.
3. Earchy, J., Quality In Use processes and their integration Part 1 - Reference Model. 2000b, Lloyd's Register: London.
4. Earchy, J., Usability Maturity Model: Processes. 1999, Lloyd's Register of Shipping.

5. Iivari, Netta – Jokela, Timo, Evaluating a Usability Capability Assessment. in NordiCHI. 2000. Stockholm.
6. Iivari, Netta – Jokela Timo, Performing A User-Centred Design Process Assessment in a Small Company: How the Company Perceived The Results. in European Software Day. Workshop on European Software Process Improvement Projects at the 26. Euromicro Conference. 2000. Maastricht, The Netherlands: Österreichische Computer Gesellschaft, Wien.
7. ISO/TR15504-2, Software Process Assessment - Part 2: A reference model for processes and process capability. 1998, International Organization for Standardization, Genève, Switzerland.
8. ISO/TR15504-3, *Software process assessment - Part 3: Performing an assessment*. 1998, International Organization for Standardization, Genève, Switzerland.
9. ISO/TR15504-4, *Software process assessment - Part 4: Guide to performing assessments*. 1998, International Organization for Standardization, Genève, Switzerland
10. ISO/TR15504-5, Software Process Assessment - Part 5: An assessment model and indicator guidance. 1998, International Organization for Standardization, Genève, Switzerland.
11. ISO/TR18529, Human-centred Lifecycle Process Descriptions. 2000, International Organization for Standardization: Genève, Switzerland.
12. ISO13407, Human-centred Design Processes for Interactive Systems. 1999, International Organization for Standardization, Genève, Switzerland.
13. Jokela, T. Usability Capability Models - Review and Analysis. in HCI 2000. 2000. Sunderland, UK.
14. Jämsä, Mikko – Abrahamsson, Pekka, Gaining staff commitment to user-centred design by performing usability assessment - key lessons learned. in NordiCHI. 2000. Stockholm.
15. Järvinen, Pertti, On Research Methods. 1999. Opinpaja Oy: Tampere.
16. March, S.T. and G.F. Smith, Design and Natural Science Research on Information Technology. *Decision Support Systems*, 1995. 15(4): p. 251-266.
17. Rosenbaum, S. What Makes Strategic Usability Fail? Lessons Learned from the Field. in CHI '99. 1999. Pittsburgh, USA.

Building an Experience Base for Software Engineering: A Report on the First CeBASE eWorkshop

Victor Basili, Roseanne Tesoriero, Patricia Costa, Mikael Lindvall, Ioana Rus,
Forrest Shull, and Marvin Zelkowitz

Fraunhofer Center for Experimental Software Engineering Maryland
{vbasili, rtesoriero, pcosta, mikli, irus,
fshull, mzelkowitz}@fc-md.umd.edu

Abstract. New information is obtained by research and disseminated by papers in conferences and journals. The synthesis of knowledge depends upon social discourse among the experts in a given domain to discuss the impact of this new information. Meetings among such experts are, however, expensive and time consuming. In this paper we discuss the organization of CeBASE, a center whose goal is the collection and dissemination of empirically-based software engineering knowledge, and the concept of the online workshop or *eWorkshop* as a way to use the Internet to minimize the needs of face-to-face meetings. We discuss the design of our eWorkshop and give the results of one eWorkshop that discussed the impact of defect reduction strategies.

1 Building an Experience Base for Software Engineering

Software development is a people- and knowledge-intensive activity; it is a rapidly changing field, and although it is slowly maturing, many activities are still ad hoc and depend upon personal experiences. In order to cope with such restrictions as firm deadlines and shrinking budgets, software-developing organizations need assistance in setting up and running increasingly critical projects.

In order to reach their goals, software development teams need to understand and choose the right models and techniques to support their projects. They must answer key questions: what is the best life-cycle process model to choose for this particular project (from waterfall to extreme programming)? What is an appropriate balance of effort between inspections and testing in a specific context? And what are the benefits, if any, to buy a readily available software component instead of developing it?

These questions are not easy to answer. In some cases the knowledge exists to answer such questions; in other cases it does not, so instead on relying on knowledge and experience, we must trust our instincts. In order to support this decision-making activity, we need to develop empirically based software models in a systematic way, covering all aspects from high-level lifecycle models to low-level techniques, in which the effects of process decisions are well understood. However, context plays an im

portant role as most projects and organizations differ. Consequently, the knowledge must be formulated relative to the development context and project goals.

The Center for Empirically-Based Software Engineering (CeBASE¹) was organized to support this goal. CeBASE will accumulate empirical models in order to provide validated guidelines for selecting techniques and models, recommend areas for research, and support software engineering education. CeBASE's objective is to transform software engineering from a fad-based practice to an engineering-based discipline in which development processes are selected based on what is known about their effects on products, through synthesis, derivation, organization, and dissemination of empirical knowledge on software development and evolution phenomenology.

CeBASE is a new National Science Foundation-sponsored research center led by personnel with extensive industry and government experience, including Professors Barry Boehm (University of Southern California), Scott Henninger (University of Nebraska Lincoln) Rayford Vaughn (Mississippi State University) and co-author Victor Basili (University of Maryland and Fraunhofer Center for Experimental Software Engineering – Maryland). Their experience includes major improvements in software productivity, quality, and cycle time on highly complex software-intensive systems. Within CeBASE we are integrating existing data, methods, and models (including the Experience Factory, Goal-Question-Metric approach, Spiral life cycle Model, Model-Based (System) Architecting and Software Engineering (MBase), stakeholder Win-Win requirements engineering, COCOMO cost and schedule modeling, and perspective-based reviewing) with the best commercial and government practices (such as the CMMI, Rational Unified Process, Independent Expert Reviews/Architecture Review Boards, SEI Product Line, Risk, and Architecture practices). The result will be tailorable methods for defining, managing, and continuously improving future software intensive systems involving challenges such as COTS based development, systems distribution and mobility, and tradeoffs among cost, schedule, performance, dependability, interoperability, usability, and other factors. The initial focus of CeBASE is on two high-leverage areas of software engineering, defect reduction and COTS based development.

CeBASE collects, documents, and disseminates knowledge on software engineering gained from experiments, case studies, observations, and real world projects. While some of this empirical knowledge might be well known by the community, it has not yet been documented. Although this knowledge is believed to be generally applicable, the effects of its application have never been systematically investigated making it difficult to discern when it is useful. Some of this knowledge is distributed among many individuals, which means that we need to gather the pieces together and facilitate the collection and management of collective knowledge.

Meetings among experts discussing their findings and recording their discussions are a classical method for creating and disseminating knowledge. By analyzing such discus-

¹ <http://www.CeBASE.org>

sions new knowledge can be created and the results can be shared. This is generally achieved by holding workshops. Workshops, however, possess limitations: 1) experts are spread all over the world and would have to travel, and 2) workshops are usually oral presentations and discussions, which are generally not captured for further analysis. To overcome these problems we designed the concept of the *eWorkshop*, using the facilities of the Internet. This paper describes the process of running an eWorkshop and the results from the first eWorkshop, which was held March 16, 2001.

2 Conducting an eWorkshop

The eWorkshop is an on-line meeting, which replaces the usual face-to-face workshop. While it uses a Web-based chat-application, it is structured to accommodate the needs of a workshop without becoming an unconstrained on-line chat discussion. The goal is to synthesize new knowledge from a group of experts as an efficient and inexpensive method in order to populate the CeBASE experience base. The idea behind the eWorkshop was to use simple collaboration tools, thus minimizing potential technical problems and decreasing the time it would take to learn the tools. Simultaneously, we wanted to set up a support team and control room to ensure that there would be as few disturbances as possible once the eWorkshop was running.

2.1 eWorkshop Process

Organization of the workshop follows a strict protocol:

1. *Choose a topic of discussion.* The topic under discussion is first determined.
2. *Invite participants.* Participants are invited and instructed to log into the eWorkshop using a Web browser at the appointed time.
3. *Distribute Pre-meeting information sheet.* To direct the discussion, preliminary information about the topic is presented to the participants, who send in a pre-meeting information sheet. This is used to guide the discussion during the meeting.
4. *Establish meeting codes – for meeting analysis.* The workshop organizers analyze the information sheets to develop a taxonomy of issues to be discussed.
5. *Publish synthesized info from pre-meeting sheets.* An analysis of the information sheets are given by the eWorkshop team and distributed to each participant before the meeting.
6. *Schedule pre-meeting training on tools.* A preliminary work session is scheduled to give meeting participants a chance to try out the software so that the meeting can proceed smoothly.
7. *Set up control room.* Several individuals (described later) actually run the meeting. While most participants are in their own offices looking at a computer screen, the meeting organizers need to coordinate their activities among several roles.
8. *Conduct meeting.* At the appointed time, the participants use their Web browser to log into the chat tool and the meeting is underway.
9. *Post-meeting analysis and synthesis.* The meeting organizers keep a script of the meeting. This is analyzed to extract knowledge for the knowledge base.

2.2 eWorkshop Roles

Most participants in an eWorkshop are experts in their respective domain. Our lead discussants (workshop leaders) formed part of the CeBASE team that interacted with an international group of invited participant experts. To minimize disturbances during the meeting and to capture important information, we relied on a support team operating from a single control room. This support team consisted of the following roles: *moderator*, *director*, *scribe*, *tech support*, and *analyst*. A phone line was set up in the control room and the number along with an email address were given to each participant to use if they had any technical problems during the meeting.

The moderator was responsible for monitoring and focusing the discussion (e.g., proposing items on which to vote) and maintaining the agenda. Of the support team, only the moderator was an active participant in the sense that he contributed actual responses during the meeting. The director was responsible for assessing and setting the pace of the discussion. He decided when it was time to redirect the discussion onto another topic. As the discussion moved from one topic to another, the scribe highlighted the current agenda item and captured and organized the results displayed on the whiteboard area of the screen. When the participants reached a consensus on a particular item through a vote, the scribe summarized and updated the whiteboard to reflect the outcome. The contents of the whiteboard became the first draft of the meeting minutes. The analyst coded the responses according to the pre-defined taxonomy. The analyst entered one or more codes to categorize responses as they were entered. The tech support was responsible for handling any problems that might occur with the tools. For example, some participants accidentally closed their sessions and had difficulty logging into the meeting for a second time. The tech support assisted these participants in troubleshooting their problems.

2.3 eWorkshop Tool

The web-based chat-application used to run the eWorkshop allows participants to create identities based upon their names. Instead of communicating statements orally, the participants submit statements by typing them and pressing the submit button. The statement appears on all participants' message boards. By responding to statements on the message board, participants can carry on a discussion online. This allows participants to be located remotely from the control room. Moreover, all statements are automatically captured in real-time, allowing them to be analyzed afterward.

The eWorkshop tool is based on web technology. The screen has five main areas: the *agenda*, *input panel*, *message board*, *whiteboard*, and *attendee list*. There are also features supporting the collaboration, such as a chat log and directory of frequently asked questions. (See [Fig. 1](#))

The *agenda* is managed by the moderator and indicates the status of the meeting (“started”, “stopped”) as well as the current item under discussion. Immediately above the agenda is a link to the pre-meeting information submitted by the participants, which is used to establish a common understanding of the participants’ status before the meeting is started. This is accessible throughout the meeting.

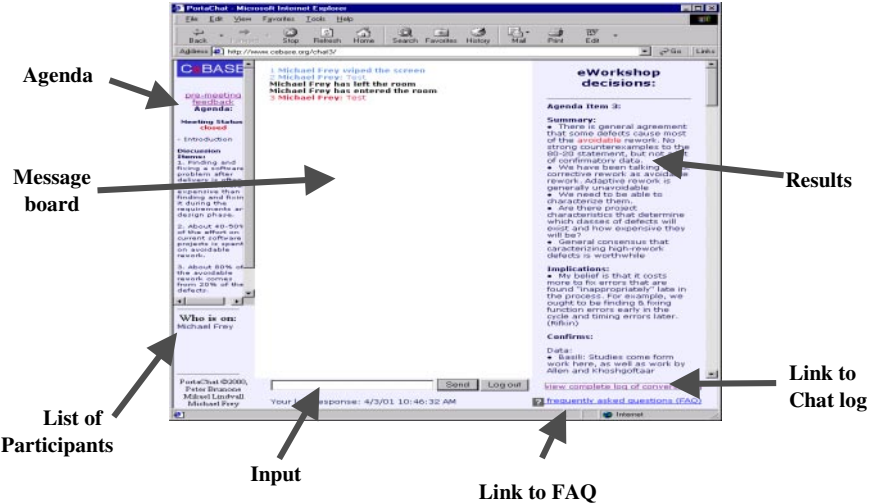


Fig. 1. Screen shot of the chat tool used for the eWorkshop

The *input panel* enables participants to type statements during the discussion. The statement appears in the text box until the *Send* button is pressed.

The *message board* forms the meeting discussion area. Each statement is tagged with a unique number in addition to the name of the participant who entered it. Statements are displayed sequentially. The number is used for later referral to a particular statement. The statement is also tagged with the time of when it was sent (this piece of information is not shown in this panel, but in the chat log).

The *whiteboard* is used to synthesize a summary of the discussion and is controlled by the scribe. In order to realize our goal of measuring the level of consensus among the participants, all of the items added to the whiteboard were subject to voting announced by the moderator. When participants did not agree with how the statements on the whiteboard were formulated negotiations were initiated in order to come up with a more accurate description of the results of the discussion.

3 Defect Reduction eWorkshop

The first eWorkshop discussed defect reduction techniques and was based on an article [2] that provided a top-ten list of empirical data that many software practitioners found very helpful. An example of a top 10 defect item is: “Finding and fixing defects after delivery is 100 times more expensive than finding and fixing during requirement and design phase.” This is an example of common knowledge that few people dispute in general, but it raises the question: “Does this knowledge always apply to all projects and under all circumstances?” In order to gain more knowledge, we decided to run eWorkshops with recognized experts on defect reduction from all over the world and collect data that supported or refuted the heuristics.

Based on this top-10 defect reduction list the goals of the eWorkshop were the following:

1. Gain a better shared understanding of key software defect reduction techniques.
2. Build a broader base of empirical results on software defect reduction to help support project decisions.
3. Conduct an assessment of the current eWorkshop’s effectiveness in strengthening empirical software engineering knowledge.

3.1 eWorkshop Preparations

Several activities based on the goals of the meeting took place prior to the actual eWorkshop. As described in Section 2.1, participants were asked to fill out and submit a pre-meeting information sheet regarding their views on the heuristics under discussion. To support the final meeting goal, we provided mechanisms to facilitate the use of the tools and established a measurement plan for evaluating the eWorkshop concept.

Content-Related Preparations

After potential participants responded to our invitation, we sent them the list of heuristics to be discussed during the eWorkshop and asked each to consider:

- If they had data that confirmed/refuted the heuristic;
- If they could refine the heuristic (e.g. by suggesting more accurate guidelines or specifying to what types of systems it applies);
- If they had other references or citations that were relevant as resources concerning this heuristic;
- If they could state the implications for practice, if this heuristic is true.

It was useful to group answers based on whether they supported or refuted the heuristic, to get a first impression of the amount of support on both sides of the issue. As in the actual eWorkshop, participants substantiated their answers with a range of different types of support, ranging from anecdotes to qualitative personal experience to references or hard data collected from a number of organizations. Rather than catego-

rize the responses based on the level of support, we merely grouped responses on different sides of the issue and let readers draw their own conclusions about the relative merits of each.

The aggregated responses² were sent to all participants to allow them to familiarize themselves with the other participants, their backgrounds, and opinions before the actual chat. The aggregated responses were also useful during the chat, as they allowed participants to refer to data they had previously submitted without re-entering them during the real-time discussion.

Tool Preparations

We asked all participants to test the meeting software to become familiar with the technology via two training sessions several days before the meeting. During that time, members of the support staff were present in the control room to answer questions and help the participants become more familiar with the tool. In addition, a list of frequently asked questions (FAQ) was placed on the CeBASE web site to explain how to use the software and identify conventions established to facilitate the discussion. Meeting participants were encouraged to read the FAQ and add additional questions, if necessary.

In order to evaluate the content of the meeting as well as the effectiveness of the eWorkshop as a forum for discussing software engineering topics, we established a data collection plan. Since the workshop was to be conducted using a chat-based tool that required participants to type each response, the actual transcript of the meeting was recorded. Additional user information such as the time of response, user name and IP address were captured and stored along with the text of each response. To obtain a better understanding of the meeting, we planned to code each participant's response according to a pre-defined set of categories based on the heuristics that were to be discussed in the meeting (content-based categories) and on aspects of the meeting itself (meeting-based categories).

Post Meeting Analysis

In evaluating the eWorkshop concept, we are interested in two main points: the effectiveness of the concept in generating new information on the topic of defect reduction and the effectiveness of the tools in supporting the meeting. During the meeting, we collected data in various forms to enable an evaluation of these points. The following list represents the data sources available for the evaluation of the eWorkshop:

- the transcript of the actual text from the meeting
- the scribe summary approved by the participants
- the analyst's coding of each captured response
- the users and IP addresses by session

² Available at <http://www.cebase.org/defectreduction/eworkshop1/premeeting.htm>.

3.2 eWorkshop Concept Knowledge

The duration of the first eWorkshop was 2 hours. During that time, 533 responses were recorded. The meeting was globally distributed with participants joining the discussion from various parts of the United States (including Maryland, Pennsylvania, Mississippi, California and Hawaii), Europe (Denmark) and Asia (Japan). The connection speeds for the participants varied as well. Several of the participants attended the meeting connected to the Internet by a phone line using a slow modem while others had higher speed connections. The varying connection speeds did not seem to affect the ability of the participants to communicate. Fig. 2 shows the distribution of the responses among the participants in the discussion during the meeting. While the lead discussants (Barry Boehm and Vic Basili) and the moderator contributed the most responses during the meeting (14%, 8%, and 11% of the total responses, respectively), 12 of the discussants contributed at least 4% (each) of the total number of responses.

Participants share of the total discussion

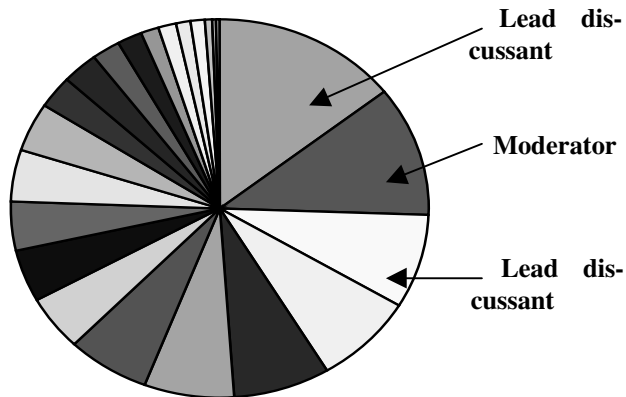


Fig. 2. Each slice of the pie represents each participant's portion of the total discussion. The two lead discussants' and the moderator's slices are marked. A majority of the meeting participants were able to add to the discussion. There was no overwhelmingly dominant voice.

The analyst's coding of the responses yielded a characterization of the content of the meeting. Most of the responses submitted by the participants were content-related. Nearly 19% of the total responses during the discussion were related to voting. 17% of the responses dealt with rework effort. 13% of the responses focused on definitions. 12% of the responses discussed actual data and citations. During the meeting, 11 citations to references in the literature regarding software defect reduction were mentioned. Approximately 10% of the discussion dealt with procedural or administrative details. This amount of overhead seems to be consistent with face-to-face meetings. For example, in [9], a study of communication effort during inspection

meetings was conducted. In the observed meetings (with average duration of 62 minutes), 14% of the communication effort was spent on administrative details.

In conducting the eWorkshop, we learned some lessons on conducting a meeting using this format:

1. There was no monopolizing voice during the meeting (unlike face-to-face meetings [11]).
2. Most of the discussion was content-related.
3. Participants need to prepare more for the meeting. Not many of the participants completed the pre-meeting information sheet and participants admitted to not preparing enough for the meeting.
4. It is important for participants to become familiar with the tool before the actual meeting.
5. The meeting seemed to run smoothly, but we did have some technical problems.
6. The simple, yet robust collaboration tools, such as the chat tool, seem to be adequate.

3.3 Defect Reduction Knowledge

In this section, we discuss the knowledge building that occurred concerning the software development heuristics that were the topic of the discussion. In this context, knowledge building generally took the form of validating whether the described heuristic held for at least some software systems; refining the heuristic by limiting the context in which it held; and suggesting implications for developers and researchers based on what is known and what data is missing.

Participants in this discussion included representatives from education:

- Ed Allen, Mississippi State University
- Victor Basili, Fraunhofer Center - Maryland and University of Maryland
- Barry Boehm, Center for Software Engineering, University of Southern California
- Winsor Brown, Center for Software Engineering, University of Southern California
- Philip Johnson, University of Hawaii
- Dan Port, Center for Software Engineering, University of Southern California
- Marvin Zelkowitz, Fraunhofer Center - Maryland and University of Maryland

from industry:

- Sunita Chulani, IBM
- Noopur Davis, Davis Systems
- Ira Forman, IBM
- Yoshihiro Matsumoto, Toshiba Software Factory, Japan
- Gary Thomas, Raytheon

and independent consultants:

- Don O'Neill, Don O'Neill Consulting
- Stan Rifkin, Masters Systems
- Otto Vinter, Independent Software Engineering Mentor, Denmark

Item 1

The first heuristic addressed the belief that finding defects early in development produces a large net savings in effort: “Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.” The discussion helped refine this heuristic (it is approximately right, for certain defect classes) and identified environments where it does not apply, or where there isn't enough data to support a judgment.

eWorkshop participants generally agreed that finding and fixing software defects after delivery is much more expensive than fixing them during early stages of development – but only for certain types of defects. A 100:1 increase in effort from early phases to post-delivery was a usable heuristic for severe defects, but for non-severe defects the effort increase was not nearly as large. This heuristic is appropriate only for certain development models: new paradigms such as extreme programming (XP) remove any kind of meaningful distinction between “early” and “late” development phases, and research has not targeted such environments yet.

General data were presented that supported an effort increase of approximately 100:1. Don O'Neill described data from IBM Rochester [6] in the pre-meeting feedback that reported an increase in effort of about 13:1 for defect slippage from code to test and a further 9:1 increase for slippage from test to field (so, a ratio of about 117:1 from code to field). Based on Yoshihiro Matsumoto's background in a software factory of 2600 IT workers, average rework time after shipment is 22.85 hours versus less than 10 minutes if the work had been done prior to shipment (a factor of 137:1). Other corroboration came from Ed Allen's experiences with a telecommunications client as well as Noopur Davis' experience.

An important distinction that emerged was that the large effort multiplier holds for *severe* defects; fixing defects with lesser impact after delivery will not increase costs appreciably.

- Barry Boehm pointed out that the 100:1 factor was about right for critical defects on large projects, illustrated, for example, by the data from the Data Analysis Center for Software [7].
- Sunita Chulani also agreed that this factor was consistent with her experience for severe defects.

For non-severe defects, the effort multiplier was not nearly as large. Otto Vinter indicated that when requirements defects are excluded, his data show an approximately 2:1 relationship between after-shipment and before-shipment debugging effort: 14 hours after release versus 7.4 hours in testing before release. Barry Boehm said that the 2:1 relationship also held for the million-line CCPDS-R project completed by

TRW for the Air Force (described by Walker Royce [8]), in which early risk resolution and well-validated modular architecting were used to reduce early defects. Marvin Zelkowitz also provided development data from NASA's Johnson Space Center, which showed that the effort multipliers associated with different defect types are different: the effort just to *find* a defect increased from

- 1.2 hours early in the project to 1.5 hours late in the project, for non-severe defects
- 1.4 hours early in the project to 3.0 hours late in the project, for severe defects.

Other variables likely to have an impact were proposed, although no supporting data was available. Gary Thomas pointed out that post-shipment costs would be expected to increase even further when a different organization than the one that developed the software maintains the system. Winsor Brown suggested that the ratio would be different for the use of a ROTS (Research Off the Shelf) system. Since such software is offered "as is", the effort spent on finding and fixing defects is minimal, and when defects *are* found and fixed, the cost is usually minimal (since only the easy ones get "fixed"). And Philip Johnson remarked that research has so far neglected development environments that do not fit into the "waterfall family" of development approaches. For example, in XP, requirements and implementation phases are so entwined that it no longer makes sense to talk about "early" vs. "late" phases of development.

Item 2

The second heuristic concerned the impact of defects on project effort: "About 40-50% of the effort on current software projects is spent on avoidable rework." Again, discussants believed this heuristic to be about right for certain types of projects – however, they helped refine the heuristic by suggesting typical rework rates for different classes of projects.

Most eWorkshop participants believed that significant amounts of effort are spent on avoidable rework. The data across many projects had however a much wider range than the proposed 40-50%; on some projects cited, for example, it was as low as 10-20%. In general, it was felt necessary to distinguish different types of software engineering processes so that we could examine the avoidable rework rates for different types of environments. Several implications for addressing high rates of rework were suggested, but the one that attracted the most agreement argued that we need to collect more data to demonstrate these problems better: when data are collected the cost of defects and the benefits of removing them early are suddenly very evident.

Several participants cited data in support of large amounts of rework on projects:

- Vic Basili said that the 40-50% claim is borne out by the Cleanroom studies at NASA Goddard's Space Flight Center [1]
- Barry Boehm pointed out that Capers Jones' books [5] have data on rework costs that indicate that the rework fraction increases with the size of the project, and can go as high as 60% for very large projects.

- Don O'Neill submitted data from the national benchmarking effort showing that the range across projects is between 20% and 80%.

There was some agreement however that higher-maturity projects spend considerably less effort on rework. Brad Clark [4] has published analyses of the effects of process maturity in which the benefits at higher levels of maturity are traced mainly to the reduced rework effort.

- Gary Thomas, using data from Raytheon, cited a range of about 10-20% avoidable rework on higher-maturity projects.
- Barry Boehm said that the better TRW projects, such as CCPDS-R, were also able to reduce rework effort down to 10-20%.

Because of this disparity between high- and low-maturity projects, Don O'Neill suggested that we should distinguish among *disciplined software engineering*, *structured software engineering*, and *ad hoc programming* and seek to associate with each a characteristic level of effort spent on avoidable rework.

In general, comparing rework costs across projects is dangerous because it can be defined in several different ways. (For one example, Vic Basili pointed out that the rework effort collected from the Software Engineering Laboratory (SEL) at NASA was measured as the effort required to make changes due to defect corrections.)

Yet there are other potential rework measures for which researchers might not even be able to collect metrics, for example the rework that is found on volatile development teams, where people are often added or removed and as a consequence spend time relearning or redoing the same things.

Some implications of high rework effort were noted. Stan Rifkin proposed that for the present, we have to expect high rework effort and use that to help projects budget effort better. Stan Rifkin, Vic Basili, and Noopur Davis concurred that collecting data is a necessity, since once people start tracking defects, the cost of defects and the benefits of early removal become very clear.

Otto Vinter proposed that the research community focus more on studying the causes of defects and finding preventions for them. Noopur Davis took this argument a step further by saying that defect prevention efforts should concentrate on removing defects as early in the lifecycle as possible; Gary Thomas suggested that formal inspections help in this regard [10]. Barry Boehm reported that at TRW it was found that early prevention effort (via reviews, inspections, and analysis tools) had a 5:1 or 10:1 payoff.

Item 3

The third heuristic concerned the contribution of various defects to the amount of rework on a project: "About 80% of the avoidable rework comes from 20% of the defects." Unlike the earlier heuristics, little existing data addressed this topic.

There was general agreement that relatively few defects tend to cause most of the *avoidable* work on software projects. (However, it was clear that there is a significant amount of additional unavoidable rework that comes from sources such as adaptive maintenance. Discussants felt that software engineering researchers need to spend more work on characterizing the types of rework and their causes.) There was little confirmatory evidence about the 80/20 rule, but there were no strong counterexamples either. In terms of the implications of this statement, there was a general consensus that characterizing high-rework defects would be worthwhile.

On this topic, little data was put forward. While most participants indicated that they believed that most of the avoidable rework comes from a small number of defects, no data from personal experience was cited. Some confirmatory data from the SEL and from the work of Khoshgoftaar and Allen was described. A dissenting voice came from Noopur Davis, who felt that the sheer number of defects that have to be found and fixed must contribute more than 80% of the avoidable rework.

Some time was spent trying to make definitions clearer. First, “rework” was defined broadly to include the effects of such things as changing operating systems, databases, or customer base; possibly also the re-configuration of tools.

Finding a broad definition of “defect” was more difficult. Winsor Brown said that this is not a unique problem during the eWorkshop: for example, in the absence of objective definitions the ODC (Orthogonal Defect Classification) researchers were forced to define a defect broadly, as any change made to the software. Stan Rifkin suggested that the CAPPU taxonomy used in the software maintenance community could help refine the definition. CAPPU characterizes changes to systems as Corrective (to mitigate the effect of errors), Adaptive (to port the system), Performance-related (to improve system performance), Preventive (to change something before it causes a problem), and User-requested (to add new functionality). The “defects” referred to in the 80/20 rule refer specifically to changes in the corrective and performance-related categories only.

Some definitions of “unavoidable rework” were presented to allow participants to agree on the types of things that were *not* covered by the agenda item. The basic idea, which seemed to have consensus, was that unavoidable rework was rework that came from other sources than defects, e.g. from Adaptive, Preventive, or User-requested changes to the code or architecture. Otto Vinter suggested expanding this definition by proposing that unavoidable rework could be caused by some defects that are simply too hard to prevent.

Most of the discussion centered on identifying what types of defects were most likely to cause disproportionately large amounts of rework. Barry Boehm said that in his experience one source of high-rework defects is “architecture-breakers:” defects whose fix requires significantly changes to the architecture, which then necessitates design and code changes.

Stan Rifkin described his belief that it costs more to fix errors that are found "inappropriately" late in the process. For example, we ought to be finding and fixing function errors early in the cycle and timing errors later, so function errors that are not found until the later stages will not cause a higher amount of rework. Barry Boehm commented that IBM ODC data (from Ram Chillarege's papers) [3] do indicate that some defects tend to be found earlier (e.g., function defects) and others tend to be found later (e.g., timing defects). Another implication of this is that timing defects are probably more expensive to fix than function defects, because they cannot be found in earlier phases where corrections would be cheaper.

Winsor Brown and Otto Vinter suggested another potential classification of defects: classifying based on potential prevention technique. Clearly some defects could have been avoided with better education, training, or information.

3.4 Participants' Feedback

The majority of the participants liked the eWorkshop. They thought it was a "good way to discuss," "worthwhile and stimulating" and "a relatively easy way for a group to get a discussion going." Some of the difficulties people reported were related to the tool, but most resulted from their lack of preparation in using the technology before the meeting.

People generally said they would participate again and would recommend this discussion vehicle to others, with the comment that more pre-meeting preparation would be a great benefit. More thorough preparations could include sending their position relative to the topics on the agenda, together with arguments to support or refute them, such as data and references.

4 Next Step

The first eWorkshop addressed the first three items of the top-ten list that focused on cost and effort. The second eWorkshop will be held on July 16, 2001 at 11:30 AM EDT. It will focus on the impact that defects have on software. These correspond to top-10 heuristics 4, 5, 9 and 10. A third meeting will discuss the remaining top-ten list items on effective methods to find defects. After the defect reduction eWorkshops, a new series of eWorkshops will be held that will discuss the top-ten list for COTS based systems.

We are currently improving the chat tool based on suggestions from the first eWorkshop and this improved version will be used for the second eWorkshop.

Although the eWorkshop idea proved so far to be an efficient approach for gathering collective knowledge, we do not believe the technology has progressed to the point

where all face-to-face meetings can be eliminated. For this reason, we are planning to have a (no "e") workshop to conclude this eWorkshop series in Spring 2002.

We are constantly updating our website, its support tools and feedback mechanisms, to facilitate interaction between CeBASE experts and the software engineering community. Please visit our website at <http://www.cebase.org> frequently for more information!

5 Conclusions

CeBASE has an ambitious goal of collecting relevant empirically-based software engineering knowledge. The eWorkshop has been proposed as a mechanism for inexpensively and efficiently capturing this information. Based upon an initial virtual meeting in March 2001, the process seems effective, and most participants were satisfied with the results. We have learned from some flaws in this initial process, however, and are evolving the concept to where we believe it can replace a significant portion of face-to-face meetings using the virtual meeting capabilities of the Internet. We are planning several more meetings over the next few months where this concept can evolve.

The results from the first eWorkshop generally confirm the first three items in the top-ten defect reduction list. We obtained additional references and data that seek to classify when specific defect reduction heuristics are applicable. We are in the process of further analysis of this data, which we will place on the CeBASE web site. Once we refine our eWorkshop capabilities, we can run eWorkshops more frequently on a wider variety of topics to help evolve the CeBASE experience base rapidly.

6 Acknowledgments

This work is partially sponsored by NSF grant CCR0086078 to the University of Maryland with subcontract to the Fraunhofer Center - Maryland.

We want to thank Barry Boehm, Scott Henninger, Rayford Vaughn, Winsor Brown, Dan Port and Michael Frey as well as all participants for their contribution to the success of the eWorkshop. We also want to thank students at USC and UMD for their contribution in testing the system and Jennifer Dix for proof reading this paper.

References

1. Basili, Victor R and Green, Scott, 'Software Process Evolution at the SEL,' IEEE Software, pp 58-66, July 1994.

2. Boehm, Barry and Basili, Victor, "Top 10 Defect Reduction Techniques," IEEE Computer, January 2001. (Also available at <http://www.cebase.org/defectreduction/top10/>.)
3. Chillarege, Ram, Bhandari, Inderpal, Chaar, Jarir, Halliday, Michael, Moebus, Diane, Ray, Bonnie and Wong, Man-Yuen, "Orthogonal defect classification – a concept for in-process measurements," IEEE Trans on Software Engineering, 18(11), November 1992: 943-956.
4. Clark, Bradford, "The Effects of Process Maturity on Software Development Effort," Ph.D. Dissertation, USC, 1997. (<http://sunset.usc.edu/reports>)
5. Jones, Capers, Applied Software Measurement, 1996.
6. Lindner, Richard J. & Tudahl, D. "Software Development at a Baldrige Winner," Proceedings of ELECTRO '94, Boston, Massachusetts, May 12, 1994, pp. 167-180.
7. McGibbon, Thomas, Software Reliability Data Summary, DACS, 1996.
8. Royce Walker, *Software Project Management: A Unified Framework*, The Addison-Wesley Object Technology Series, 1998, Appendix D.
9. Seaman C. B. and Basili V.R., "Communication and Organization: An Empirical Study of Discussion in Inspection Meetings," IEEE Transactions on Software Engineering, 24(6), June 1998.
10. Don O'Neill, "Peer reviews and software inspections," Encyclopedia of Software Engineering, Wiley Publishing. To be published. Draft can be found at <http://members.aol.com/ONeillDon2/peer-reviews.html>.
11. "Fundamental Skills Course" GroupSystems.com, 1999

Experience Magnets

Attracting Experiences, Not Just Storing Them

Kurt Schneider

DaimlerChrysler AG, Research Center Ulm
P.O. Box 2360, 89013 Ulm, Germany
Kurt.Schneider@DaimlerChrysler.com

Abstract. In a large company like DaimlerChrysler, learning from the experiences of others is crucial for many software tasks. One tool to assist experiential learning is the so-called Experience Base. It has traditionally been seen as a mere storage and administration device for experience packages. At best, those packages were annotated to allow searching along ontologies or in a case-based way. There are, however, several ideas to go beyond mere administration and storage. Two approaches are sketched in this paper that try to use more of the intrinsic power of a computer-based tool: one is an approach to capture design rationale while software prototypes are demonstrated. The other stems from an on-going project to enhance interaction of a Community of Practice that is channeled through an Experience Base. Both examples are explained as elements of an Experience Base that actively attract experiences instead of passively storing them.

1 Introduction

Many modern systems heavily rely on software. Premier cars, for example, contain a significant amount of software to implement value-adding functionality, such as the Electronic Stability Package (ESP), Airbag, or Intelligent Braking Systems. There is also a lot of software to support business processes, from Internet ordering to remote diagnosis systems.

In a large company like DaimlerChrysler, a growing workforce is involved in software development, acquisition, and integration. During the current shortage of computer science graduates, there is an equally growing demand for experienced software professionals. In order to meet this demand, e.g. in quality assurance and quality management, several approaches have been taken. External software quality experts are hired, and employees with a different background are reassigned to software quality tasks.

Quality agents are expected to

- plan, schedule, check and monitor software quality activities;
- motivate, train, and support all quality-related activities and to explain to all project participants why they need to be performed;

- balance software quality demands with project budget constraints and schedule, thus making reasonable trade-offs and compromises.

At the same time, software quality people are often the *only* advocates of software quality left as soon as a project slips into troubled water. There is rarely more than one explicit quality agent in a project. This fact even increases quality agents' need for experience exchange across projects to support them when projects get tough. Quality experts need good communication and conflict management skills [1]. And most of all, they need the experience that allows them to make informed decisions and appropriate trade-offs when schedule, budget and quality demands collide. Newcomers lack this experience.

Some disciplines have a special need for experience-based support:

- when there are no clear success criteria and
- when technical expertise must be combined with a human capability for judgement and compromise.

Software requirements, quality, and design are some fields that match this description.

At DaimlerChrysler Research and Technology, we have our so-called "Software Experience Center (SEC)" project running since 1998 [2]. Its purpose is to initiate, seed [3], and support experience-based software process improvement in participating business units. Software quality is one domain we are focusing on.

In section 2, Basili's seminal work on Experience Factory [4] is cited as our anchor for the Software Experience Center. This paper focuses on the "storage device" of such an organization, which Basili called the "Experience Base". However, Experience Bases have long remained passive.

The key claim of this paper is: Experience Bases have so far exploited only a small portion of their potential as computational tools. Section 3 analyzes what could be done in principle to go beyond storing, searching, and posting. In essence, collecting experience should be turned from an annoying, effort-consuming exercise to the by-product of doing other meaningful work. Speaking metaphorically, experiences should be attracted to Experience Bases like iron to a magnet. Sections 4 and 5 explain two approaches that have both been implemented, but not yet (fully) integrated into the Experience Bases that operate in our business units. Both cases illustrate the concept of "Experience Magnets".

2 Traditional Experience Base Concepts Related Work

The Origin: Experience Factory

Basili's original work in this context [4], [5] does not go into any detail about the organization, context, or use cases of an Experience Base. An Experience Factory is an organizational unit that collects, compares and engineers experiences and data from several projects, and for the benefit of several (other) projects. Experience Bases

are to be used in this context of goal-oriented, measurement-based software project improvement. Measurement planning is operationalized through the Goal-Question-Metric paradigm [6]. As a consequence, the first Experience Factory at NASA [7] mainly stored measurement results that had been analyzed and interpreted (also using GQM). Obviously, explanatory texts (such as technical reports) need to complement measurement data.

Qualitative Experience Bases

Our own work started with measurement-oriented Experience Bases, too [8],[9]. After initial attempts, most business units demanded different experience-based information to complement measurement data. In particular, a GQM measurement program was often perceived as far too time and effort consuming. Qualitative pieces of experience seemed to be a natural complement to the highly condensed quantitative measurement results. It was important to know what had worked, and what not. Tips and tricks were in high demand. The differences of our Software Experience Center to the original NASA Experience Factory concept are described in [2].

When Experience Bases are organized like traditional databases, this reveals another, more subtle mismatch: there is a difference between what authors tend to store (e.g. experiences reports) and what potential users need in their daily work (e.g., immediate support, best practices). Since users are business unit professionals (e.g. software quality agents in a project), they are driven by concrete demands for advice and instruction, not an academic interest in others' experiences. Other companies report the same observation [10], [11]. When we carefully studied the real "use cases" of our target users, we found that Experience Bases needed to offer more specialized, and more sophisticated support than just storing experiences and allowing people to check them out exactly as they had come in. Several collected pieces of experience need to be combined, analyzed and synthesized to more reusable support material.

The process of analyzing and "engineering experiences" has always been carried out off-line: So far, Experience Bases have only be used to store incoming experiences like in a database, and to distribute (or more precisely: post) refined and engineered experiences once they were transformed into reusable best practices.

Influence of Knowledge Management

Sophisticated search machines or mechanisms are often the only attempt to assist users in getting out of an Experience Base more than what authors have punched in. Two approaches have tried to apply powerful mechanisms to the structuring and searching of pieces of experiences: ontologies [12] and case-based reasoning [13] are both techniques rooted in artificial intelligence. They have been applied in other fields, and Experience Bases are only one additional application domain.

Both approaches are appropriate for large collections of many objects. Large collections need a mechanism to sustain consistency of vocabulary. However, those benefits come at a high prize: We tried ontologies and found the effort to attribute

each and every object a major acceptance hurdle [14]. We expect progress in (1) semi-automatic indexing (filling attributes) for both approaches to reduce effort; and (2) co-evolving ontologies that grow with the software discipline they model. Static ontologies are inadequate for a dynamic application domain like experiences about software processes. Our own research is devoted to other, additional approaches of making experience exploitation more effective and more efficient.

Optimizing Experience Capturing for Later Reuse

What may seem obvious in hindsight, first was an eye-opener:

- re-users of experiences often want other things than what experienced authors tend to document; and
- experience documentation often is a pain in the neck for experienced experts.

To face this challenge, we tried to develop techniques and tools to allow capturing experiences in a way that would not be too labor-intensive for those who contributed. The results should be made more reusable. Successful examples of fine-tuned techniques are

- *Observation Packages* [9]: one-page forms to capture experiences while one is still in action. Technique has been optimized in many ways, by asking the kind of questions people are willing and able to answer. The questions guide experienced experts to report an observation, reflect on their emotional reaction, and to explicitly conclude from it. For example, we collected over forty completed "observation packages" on software inspection. They are immediately useful for readers, and they can easily be analyzed for improving best practices.
- the *LIDs technique* [15] is optimized to capture experiences from a small group of co-workers who were involved in a common task for about three months (e.g., the test phase of a project). LIDs contains a process and a pre-defined table of contents for the resulting experience document. Again, the table of contents acts as a checklist and guides the group through the period they want to reflect about. Related documents and plans are saved and hyper-linked to the experience reported in the LIDs session. LIDs is described in [15] in detail.

There are other examples [16], some are inspired by the "reflection-in-action" slogan [17]. They all

- lower the threshold for those who are supposed to contribute experience;
- provide a structure of the expected result that helps future readers to understand the reported experiences – and, thus, to apply them.

Note that neither technology nor Experience Bases play a major role in either of the above two techniques. The computational power that is always present *anyway* when an Experience Base is invoked, is hardly touched in all these cases.

3 Exploiting Computational Opportunities

Experience Bases in a Software Experience Center are not intended to contain every imaginable kind of experience: In contrast to most knowledge management efforts [12], they are specifically devoted to only one domain: to software. It is, therefore, worth-while examining the relationship and the differences between *general* knowledge management and specific Software Experience.

General or Powerful Support: Choose *One*!

In principle, the goals of a Software Experience Center can be mapped to those of Knowledge Management. Making experiences (an essential class of knowledge) available to others, is at the core of both. Both are aware of the existence and importance of tacit [18] or implicit [19] knowledge. In both approaches, not only explicit but also implicit experience/knowledge is handled, and both approaches call for an organizational setup to integrate the capturing, analysis, and dissemination steps of a cyclic process into the structure of a business unit [20],[19]. Motivation is a key challenge in both approaches (see the Siemens *knowledge&more* incentive system, for instance [11]). Nevertheless, there are good reasons for the Software Experience Center to focus on the software domain only.

In many cases, the more specific a task or domain is specified, the more powerful a respective tool can be. For example, a fish knife is superior to a general knife when confronted with a trout. Along the same lines, we put all our energy into making SEC powerful for *software*. We did not have to consider more general applicability (e.g. to wiper construction or sales). DaimlerChrysler does have its own approach to knowledge management *in general*: there is the whole family of TechClubs and EBOKS [21] for the entire company. SEC, however, is devoted to software specifically.

Specific Benefits of Software-Specific Support

What are the benefits of a specific focus on software processes and products?

- Software products reside on a computer *anyway*, and so does an Experience Base. Ostwald [22] demonstrates the benefit of relating and (hyper-)linking an object of interest (here: software) with rationale, and experience *about* that same object.
- It is easy to record what happens on the screen. There are inexpensive tools for screen/audio microphone recording and replay.
- Computerized Experience Bases can act as storage and dissemination platforms, as pointed out in the previous section (LIDs, observation packages).
- In a computer, massive computational power can be applied to analyze structures, link between software objects, and their hyper-linked documentation [22].
- In knowledge management, users (sales agents or repair technicians) are often not familiar with computers so much [11]. In the software realm, users who spend their professional lives developing software have a much lower threshold in using a computerized Experience Base - and using non-basic functionality, too.

We show two examples that we have experimented with and discuss their relevance for a next-generation Experience Base that exploits the computational potential more.

4 Case One: Capturing Rationale and Experiences around Software Prototypes

Prototypes and Their Developers: A System

Software prototypes are a very specific example in which a software object (the prototype) is of very little value by itself: instead, it is the concepts and rationale built into the prototype, the experiences made, and the reactions received from others that make a prototype valuable. Unfortunately, very little of this valuable information ever gets documented. This situation turns into a problem when the prototype-developer system is cut apart. Then it shows that

- the prototype is not comprehensible by itself - its mission, key accomplishments and all experiences related with the prototype cannot be reconstructed;
- the developer needs the prototype to make his or her point about a concept. Rationale loses its context; design decisions drift into the abstract; and experiences with the prototype lose their object of reference.

And there is a number of reasons why prototypes and developers really get divorced:

- The developer quits or changes assignments; a student finishes a thesis; a pre-development department considers its task completed and starts the next project.
- The program language of the prototype becomes obsolete or is abandoned by the company; a new compiler version might not support some of the specific inventions in the prototype; or an environmental parameter (bandwidth, protocol, devices, etc.) may change and invalidate some (often implicit) assumptions.

During years of industrial experience, I have encountered several occasions, in which "prototypes" of any kind [23], [24] suffered such a destiny in a corporate environment. Obviously, pre-development or research departments run into the problem almost regularly, while business units experience it less frequently.

An approach for Capturing the Essence

If prototypes are to be treated as experience assets [23], one needs to conserve the value of the prototype-developer system beyond its separation. The so-called FOCUS approach [23] is mainly characterized by the following constraints and observations:

- To conserve developer knowledge and experience it must be made explicit and it must be captured.
- Asking the developer to document a prototype (without substantial support) is ineffective, since prototyping as a concept almost excludes documentation.
- Any approach that asks one person (the developer) to do work for the benefit of others (the users of the experience) will face stiff challenges – and often fail [25].

Although the above warning is easy to believe in the abstract, putting it into practice is difficult. Both our observation packages and the LIDs technique have put substantial effort in a balanced relationship. A key concept seems to lure behind phrases like "as a by-product" or "what they did *anyway*".

Capturing Experiences during Prototype Demos

One situation in which a lot of the experience and rationale is verbalized *anyway* is during a demonstration of the prototype. Prototypes are built to try out a technical solution, or to demonstrate how a concept could be implemented [24],[23].

In a good demonstration, the developer will explain what aspect the prototype is supposed to clarify; and there will be explanations on the results and experiences made with the prototype. This is exactly the essence of what is interesting about a prototype. A demonstration is by nature constrained in time; therefore, there is a natural focus on only the essence, while all the scaffolding will be ignored. Separating essence from scaffolding is the first important - implicit - step towards effective reuse.

FOCUS provides a checklist that helps developers to mention some of those questions that should be answered in every good demonstration (like: What is this? Who is supposed to use it? What is special about it? What is the essence?). This checklist for the developer helps to focus the demonstration. The next challenge is to conserve the surfaced insights.

FOCUS recommends recording the demonstration in a three-fold way:

- Recording what happened on the screen and what has been said (screen and audio capture). There are affordable COTS products that perform this task (around 100 MB per hour in a reasonable quality).
- Recording what parts of the prototype were executed during the demonstration (execution path). The most appropriate granularity for paths seem to be a sequence of object method invocations (not single statements or sub-programs) [23]. To collect those traces or execution paths, the code needs to be instrumented.
- Recording what parts of the prototype code the developer explains (if at all). Similar to the execution paths, explanation paths consist of sequences of methods that were explained. Code is displayed and explained in editors, so editors have to be "instrumented" in order to produce explanation paths.

FOCUS Process

A typical FOCUS-supported prototype demo will unfold as follows:

1. According to the FOCUS checklist, the developer will **introduce the prototype**, its purpose, and key achievements. This will already be audio-recorded.
2. Then there is the **demo proper** in which the developer shows the highlights that justify building the prototype, and the parts that are of special interest. This part will be audio and screen recorded, and an execution path will be recorded.

3. For pure demonstration prototypes that do not examine any "How to?" technical questions, the *demo might be over* at this point. The result is a well-structured record of exactly what was said and shown.
4. If there are aspects of **How a certain behavior was achieved**, the developer will now open an (instrumented) editor. One can now follow a recorded execution path; this path will then "remote control" the editor to step through and display object methods in the sequence they were executed during the demo.
5. Or the developer decides to select the methods in the editor himself or herself. The selected methods are explained.
6. In both cases, an audio/screen record of the explanations will be taken; in addition, an explanation path will be recorded by the editor.
7. After this, the demonstration is *usually finished*. There is, however, the opportunity to use some more of the computational power available.
8. A set of so-called **detectors** can be invoked. They are pattern matchers on a set of related paths. For example, all execution paths can be compared. Hot spots that are executed in most demos can be identified. Detectors ask users to compare the explanations given about the same hot spot. Inconsistencies can be fed back to the developer, such asking very specific, time-saving questions. Note that users are beneficiaries, so it is them who should be doing the work.
9. Path similarity metrics can be used to **find the most "similar" explanation path** for a given "execution path". When users later view a recorded demo, its accompanying execution path can be used to identify an equally recorded "related explanation". This is helpful once the developer is no longer available.

There are many detectors imaginable: Why is a method explained but not executed? Why is one method executed but a super class method gets explained? Why do explanation and execution paths always diverge at a certain point? It is the intention of FOCUS to help the user and the developer to efficiently *focus* on the most rewarding questions. FOCUS itself does not answer a single of the questions it asks (even though all detectors are based on heuristics of what reason might be behind a detected pattern).

Assigning Appropriate Roles

FOCUS is optimized to use computational power where humans do not want to waste their time: comparing large amounts of (path) data, matching patterns, and recording high volumes of data. Users as beneficiaries are called to do as much of the work as they can. And developers as those "giving" information are given back the feeling of "getting" social recognition, expert status, with all care taken to save their time and effort. Fischer [26], among others, warns not to confuse the specific abilities and restrictions of humans vs. machines. FOCUS operationalizes this claim in a specific technique for experience elicitation.

Analysis of FOCUS

FOCUS was originally implemented in Smalltalk. Fig. 1 shows the core elements needed: A browser for instrumented code (execution paths). A simple explanation editor that can be "remote controlled" by execution paths, and which collects explanation paths. In parallel, the standard Smalltalk Browser was extended in functionality and also "remote controlled" to display methods in the way software developers are used to. In the background, there is a commercial system which records the audio and screen activities. Detectors present their results in a text window (not shown here).

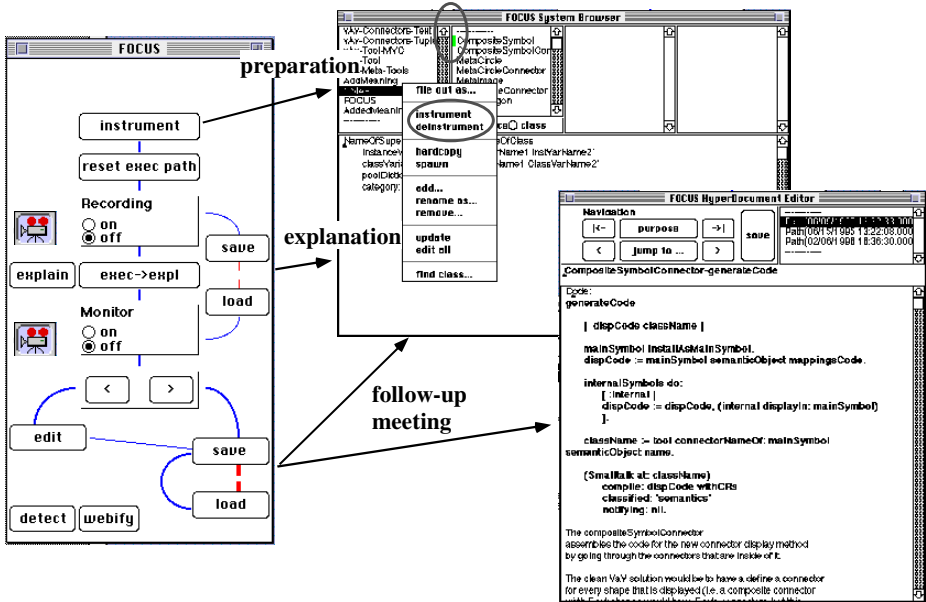


Fig. 1. FOCUS toolset of instrumented editors, compiler, and FOCUS side panel

FOCUS gains when being integrated with other experience-based tools or environments. In order to protect against the case in which a prototype "becomes inapplicable", there was a feature to "webify" all paths and selected screen shots. The audio/screen record of 100 MB/hour is only an option for local usage, not over the Web yet. FOCUS has also been coupled with a Group Memory system [27]. This was the first attempt to combine the original demonstration with emails discussing that demonstration. A data volume as high as this may seem problematic at first glance. However, storing 100 MB costs less than 25 Euro on a transportable medium; far less on a CD. Even if distribution over the Web of the full records is not affordable today, a set of CDs could be distributed and coupled with a Web-based Experience Base.

Today, FOCUS would be implemented in Java. The newest version of our Experience Bases resides in the Intranet.

5 Case Two: Collaborative Learning on Demand

Implicit Experience Exchange during Collaborative Work

FOCUS (in Case 1) dealt with making experiences explicit: the prototype developer is assisted in explaining and demonstrating essential aspects of a prototype. The prototype is an "object of interest" that resides in the computer *anyway*. This fact allows FOCUS to exploit some of the computational power to elicit experiences efficiently and effectively. However, tacit or implicit knowledge and experiences sometimes play an even more important role [19]. In the second example of a computationally empowered Experience Base, several people have some experience in a common field of practice. The challenge is for them to collaborate and come to a conclusion that combines insights from several participants. In the EU-funded CORONET project, this setting is called "Collaborative Learning on Demand". All participants learn while they work, and they work while they learn from each other.

An example application we have singled out at DaimlerChrysler to study Collaborative Learning on Demand is risk management in software projects. Risk management is a systematic and disciplined approach to deal with risks [28, 29]. One of the core activities is the analysis and prioritization of risks. In our variant of the RiskIt [28] method, prioritization is carried out using a portfolio. Risk management includes more steps, but in the context of this paper, the portfolio is the focus.

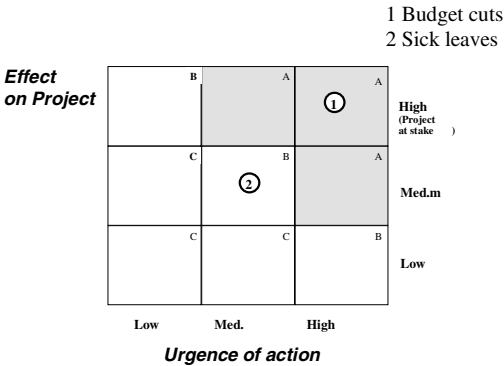


Fig. 2. Risk portfolio to prioritize risks

Prioritizing all the risks that can occur in a project is an intrinsically collaborative task. No single participant knows about all risks and influences. Such a situation requires mutual learning, which is characteristic of the disciplines that have been called experience-prone above (“combining technical expertise with creative and evaluative skills”).

As opposed to Case 1 (FOCUS), our risk management method is not performed in a computer today. There is no "object-of-interest" that lends itself to attracting experiences into a Risk Management Experience Base (Figure 3).

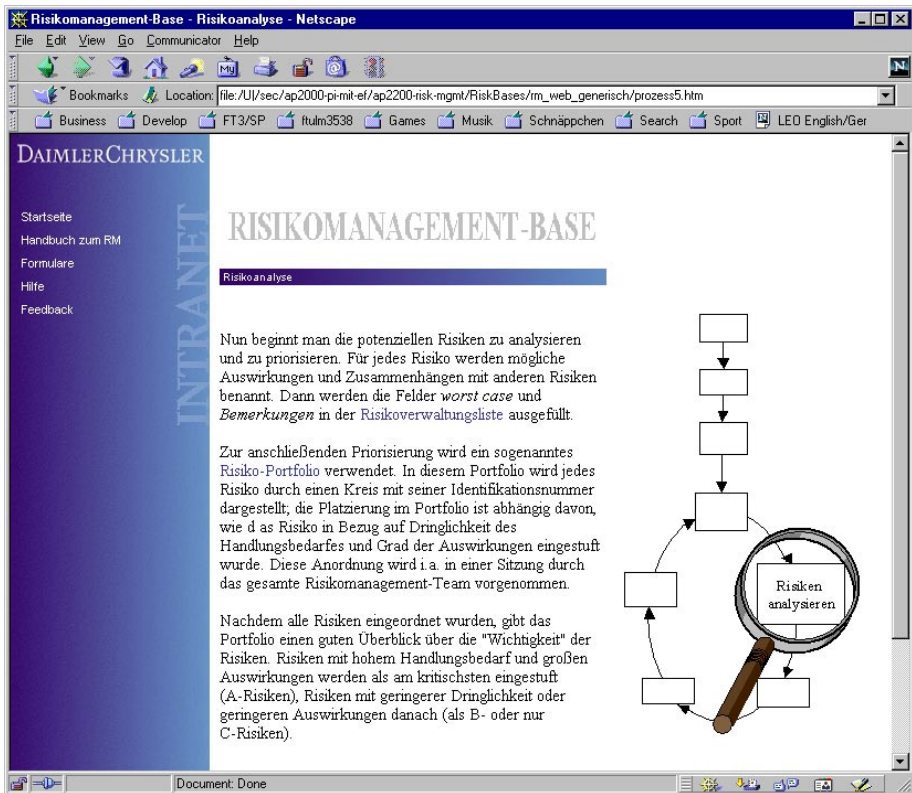


Fig. 3. Risk management process in Experience Base (analysis step highlighted; in German)

Attracting Experiences into the Computer

If the working object (like the risk portfolio) does not reside in the computer in the first place, one can try to establish focal points of a computer-based collaboration. A risk portfolio as shown in Figure 2 is a good candidate:

- it is simple to use and to understand;
- it is easily mapped on a computer, and discussions about it can be routed through a computerized communication channel;
- several stakeholders need to collaborate anyway;
- since portfolio sessions can be short, participants do not always find it time-efficient to get together for this task alone. Through making it collaborative over the Intranet, even the original task itself (of risk prioritization) is facilitated, not only its elicitation or recording.

We wanted to attract collaboration into the computer by providing a shared work space for risk prioritization.

- From the Risk Management process step in our Experience Base one can access a shared portfolio. A group of people can place and move circles that represent risks. IDs refer to a risk list that is also administered by the shared work space.
- There is a chat element synchronized with the shared portfolio (see Figure 4, left). Stakeholders discuss the relative importance of risks through the chat, and they conclude by moving risk circles.
- So far, only the *communicative* power of a computational environment would have been exploited. But in the next step, these elements are cross-bred with recording, just like in FOCUS. Risk circle movements and chat contributions are recorded in an event-based way. Both channels are synchronized.
- Recordings can be replayed in a step-by-step mode or in a quasi-continuous mode. Replay is operated through controls shown in the bottom of Figure 4.

Currently, we have specified such a computationally enriched risk management tool, and a prototype is currently being developed.

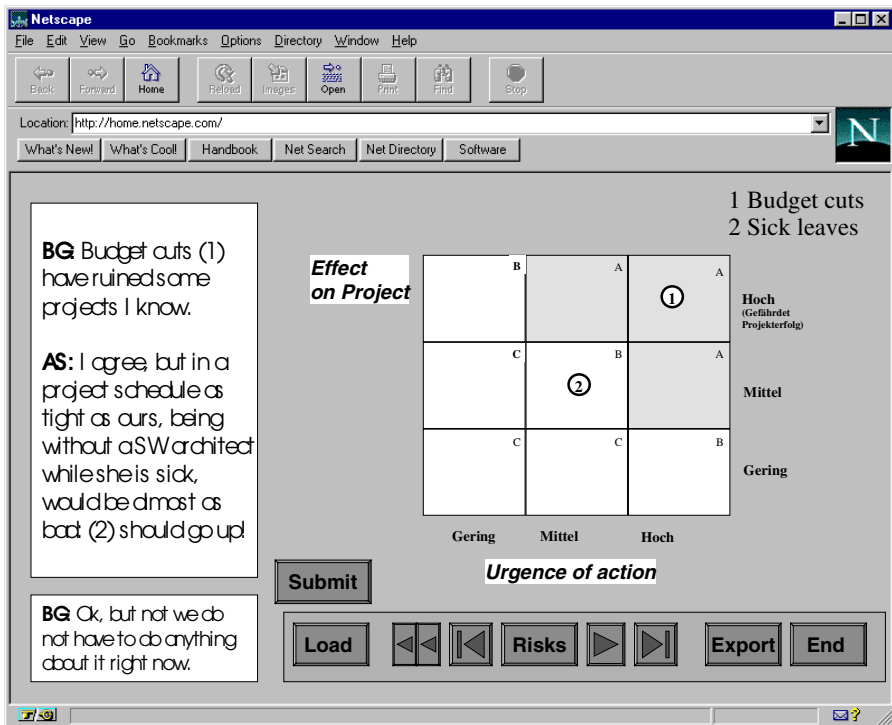


Fig. 4. Mockup of the CORONET risk portfolio

We expect to attract several people into the Experience Base by providing an environment in which they can perform risk analysis remotely. As a by-product, their discussions get recorded. This helps colleagues to catch up when they missed a meeting; it helps new colleagues to understand the arguments; and it helps risk

management novices to learn how the method is performed in practice. Experience Bases that support experience-prone collaborative tasks (like risk analysis) attract valuable experiences. They act as Experience Magnets by applying some computational power beyond information storing and dissemination. In the future, applying pattern detectors on risk discussions (like in FOCUS) will be possible.

6 Discussion

Experience Bases are the reference point for collected and refined experiences. In the realm of software, objects of interest and evolving artifacts [22] can be combined with the rationale, experiences, and insights concerning those objects. The two presented cases, FOCUS and CORONET, have several aspects in common:

Aspect	FOCUS	Coronet
A task that needs to be carried out anyway is used for experience elicitation as a by-product	Demonstration of prototype	Risk prioritization
The task at hand is partially performed with the assistance of a computer	Prototype runs on computer	Web-collaboration
This same computer is also employed to enhance the original task with recording and replay features.	Demonstration guided by previous records	Coronet enables remote risk analysis
Recording goes deeper than the screen: internal structures are the bases for recording	Execution and explanation paths	Synchronized chat contributions and risk re-prioritization
Recorded structures are analyzed and pattern can be searched to focus human attention on the most interesting points.	Execution and explanation paths through prototype code	Risk analysis session: chat contributions and risk circle movements
Roles are carefully designed and assigned to participants and the computer to let every one do what he or she (or it) can do best.	Prototype developer vs. Audience/novice/follow-up developer	Risk management team; students of risk management

There are also some differences: FOCUS starts with an object that resides in the computer anyway, whereas CORONET needs to map a previously manual task of risk prioritization to the computer. By connecting participants in a new way, the new collaborative mode of performing risk prioritization stays attractive and resides in the computer, too.

In CORONET, the connection to a risk management Experience Base is obvious: From the description of the risk analysis step in the process, there is a link to the collaborative risk prioritization. Recorded sessions can be invoked from that same process step explanation as "examples". In FOCUS, it depends on the subject of the prototype whether there is an existing Experience Base. If there is one, recorded demonstrations are an ideal illustration.

The two examples discussed here represent a whole class of applications. The above table gives an overview and can be used as a starting point to develop new techniques. Of course, there will be more than the two available right now. The WESPI ("WEB-based Software Process Improvement" [16]), is another example.

7 Conclusion

This paper argues for applying the computational power that is present with all Experience Bases. The intention is to turn passive repositories into "experience magnets" that actively attract experiences (and re-users of those experiences).

With experience magnets:

- Developers and experienced experts are attracted to the Experience Base by a task they would have to do anyway; maybe this task is even facilitated in the first place.
- Experience elicitation is easier through recording on three levels.
- Potential re-users are attracted by the amount of information recorded; by the additional, highly differentiated analyses available; by the ability to replay, pause, and compare different explanations.

To substantiate this concept, the paper presents two innovative concepts for exploiting the computational potential of software Experience Bases beyond the mechanisms of general knowledge management. More mechanisms can be invented that resemble the two cases presented above: Experience Magnets *can* be built!

Acknowledgements

The work on Coronet (Case Two) is funded by the European Commission under Contract Number IST-1999-11634, and by DaimlerChrysler in its Software Experience Center project. Michael Stupperich, Thilo Schwinn and the other members of the Coronet project have contributed to the risk mgmt. collaboration tool.

References

1. Schneider, K. *Qualitätsmanager/in: Wunschprofil und Erfahrungsaufbau*. in *SQM 2001 congress*. 2001. Bonn, Germany: SQS AG.
2. Houdek, F. and K. Schneider, *Software Experience Center. The Evolution of the Experience Factory Concept.*, in *International NASA-SEL Workshop*. 1999.
3. Fischer, G., *Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments*. Automated Software Engineering, 1998. 5(4): p. 447-464.
4. Basili, V., G. Caldiera, and D.H. Rombach, *The Experience Factory*. Encyclopedia of Software Engineering. 1994: John Wiley and Sons.
5. Basili, V. and F. McGarry. *The experience factory: how to build and run one*. in *ICSE 18*. 1996.
6. Basili, V., G. Caldiera, and H. Rombach, *Goal question metric paradigm*, in *Encyclopedia of Software Engineering*, J.J. Marciniak, Editor. 1994, John Wiley & Sons: New York. p. 528-532.
7. Basili, V.R., et al. *The Software Engineering Laboratory - An operational Software Experience Factory*. in *14th Interna. Con. on Software Engineering (ICSE'92)*. 1992.
8. Basili, V. and G. Caldiera, *Improve software quality by using knowledge and experience*. 1995, Fall: Sloan Management Review. 55-64.

9. Kempter, H. and F. Leippert. *Systematic Software Quality Improvement by Goal-Oriented Measurement and Explicit Reuse of Experience knowledge*. in *BMBF-Statusseminar 1996*. 1996: DLR (German Center for Aerospace).
10. Johansson, C., P. Hall, and M. Coquard. *Talk to Paula and Peter - They are Experienced*. in *International Conference on Software Engineering and Knowledge Engineering (SEKE'99). Workshop on Learning Software Organizations*. 1999. Kaiserslautern, Germany: Springer.
11. Davenport, T.G.P., *Knowledge Management Case Book - Best Practises*. 2000, München, Germany: Publicis MCD, John Wiley & Sons. 260.
12. Fensel, D., et al. *Ontobroker or How to enable intelligent access to the WWW*. in *11th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'98)*. 1998. Banff, Canada.
13. Tautz, C., Althoff, K.-D., Nick, M. *A Case-Based Reasoning Approach for Managing Qualitative Experience*. in *17th National Conference on AI (AAAI-00). Workshop on Intelligent Lessons Learned Systems*. 2000.
14. Steiner, W., *Ontologie-basiertes Management von Erfahrungswissen bei der Softwareprozeß-Verbesserung: Ontologie-Annotierung von Erfahrungspaketen*, in *Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)*. 2000, Universität Karlsruhe: Karlsruhe.
15. Schneider, K. *LIDs: A Light-Weight Approach to Experience Elicitation and Reuse*. in *Proc. of the PROFES 2000 Conference*. Oulu, Finland: Springer.
16. Hunnius, J.-P.v. *WESPI - Web Supported Process Improvement*. in *2nd Workshop on Learning Software Organizations*. 2000. Oulu, Finland.
17. Fischer, G., *Turning Breakdowns into Opportunities for Creativity*. *Knowledge-Based Systems*, 1994, 7(4): p. 221-232.
18. Polanyi, M., *The Tacit Dimension*. 1966, Garden City, NY: Doubleday.
19. Nonaka, I. and T. Hirota, *The Knowledge-Creating Company*. 17 ed. 1995: Oxford University Press.
20. Houdek, F., K. Schneider, and E. Wieser. *Establishing Experience Factories at Daimler-Benz*. in *Internat. Conf. on Software Engineering (ICSE-20)*. 1998. Kyoto.
21. APQC, *Building and Sustaining Communities of Practice*. Knowledge Management. 2000: APQC (American Productivity and Quality Center).
22. Ostwald, J., *The Evolving Artifact Approach: Knowledge Construction in Collaborative Software Development*, in *Department of Computer Science*. 1995, University of Colorado: Boulder.
23. Schneider, K. *Prototypes as Assets, not Toys. Why and How to Extract Knowledge from Prototypes*. in *Proc. of ICSE-18*, 1996. Berlin, Germany.
24. Lichter, H.S.-H., M.; Züllighoven, H. *Prototyping in Industrial Software Projects - bridging the gap between theory and practice*. in *Int. Conference on Software Engineering (ICSE-15)*. 1993: IEEE Computer Society Press.
25. Grudin, J. *Social evaluation of the user interface: Who does the work and who gets the benefit*. in *INTERACT'87. IFIP Conf. on Human Computer Interaction*. 1987. Stuttgart, Germany.
26. Fischer, G. and K. Nakakoji, *Beyond the macho approach of artificial intelligence: Empower human designers - Do not replace them*. *Knowledge-Based Systems*, 1992.
27. Lindstaedt, S., *Group memories: A knowledge medium for communities of interest*, in *University of Colorado*. 1998: Boulder.
28. Kontio, J., G. Getto, and D. Landes. *Experiences in improving risk management processes using the concepts of the Riskit method*. in *6th International Symposium on the Foundations of Software Engineering (FSE-6)*. 1998.
29. Hall, E.M., *Managing Risk: Methods for Software Systems Development*. 1997, Reading, MA: Addison-Wesley.

Improving Knowledge Management in Software Reuse Process

Timo Kucza, Minna Nättinen, and Päivi Parviainen

VTT Electronics, PL 1100, 90571 Oulu, Finland

{Timo.Kucza, Minna.Nattinen, Paivi.Parviainen}@vtt.fi

Abstract. We demonstrate the need for reuse in today's software development organisations in the light of widely accepted literature. Based on this need, we introduce the reuse processes and identify the knowledge dependent processes involved. We provide an introduction to the KM process model developed at VTT and describe the process of continuous improvement integrated to it. We establish a link between reuse and KM and then develop a theory of KM process improvement. Finally we give an outlook on a coming customer project in which we are going to evaluate this theory in an industrial case.

1 Introduction

Knowledge is essential in everyday work. Everyone learns by experience and this knowledge is later reused in similar kind of tasks again by adapting this knowledge to a new situation. The general purpose of Knowledge Management (KM) is to make this kind of knowledge usable for more than one individual, e.g. for an organisation as a whole, i.e. to share it. Techniques and practices similar or even identical to KM have been used for a long time, although they have neither been called by this name nor necessarily recognised as knowledge management until a few years ago [1]. Due to the steadily increasing speed at which new techniques are being developed along with the need of integrating knowledge into processes and products, knowledge is, at present, more and more widely considered as the most important asset of organisation [2]. Various industrial projects have proven that KM has potential to remarkably improve processes and products [3].

Contrary to prior assumptions, KM concerns all organisations [4]. The field of software development is especially affected, because of the continuing struggle to stay abreast of new technologies, to deal with frequently changing customer requirements and more and more complex software products, and to cope with the competition in highly demanding markets. Some issues can be addressed by distributing organisations geographically or even organisationally, but with these new management techniques also new problems arise. Many software development organisations have begun to understand that if they are to succeed in the future, they need to manage and use more effective, productive, and innovative knowledge on individual, team and organisational levels [1, 5]. Software Process Improvement (SPI) is the most general method used for meeting these changing challenges in software

development, and a large amount of profitable research and practical work has been done in this area [6, 7]. A promising, but not yet fully exploited method of improving the process of software development can be seen in the implementation of reuse [8].

The purpose of this paper is to propose an approach to applying KM to software reuse process improvement. While software itself is knowledge intensive [9], reuse has been chosen as the application domain here, because it amplifies the need for proper mediation of knowledge not only between people but also over time. The concepts KM and reuse are similar to a great extent - while reuse commonly stands for the reuse of software components, KM denotes the reuse of knowledge. The difference lies in the scope and approach of the concepts. Whereas KM has a wide scope, as almost everything can be regarded as being based on knowledge, reuse is more focused on components appearing in software development. While KM puts an emphasis on tacit knowledge, reuse focuses on explicit knowledge as embedded in reusable components. We intend to show that the combination of the two concepts can bring the underlying idea of using already existing assets to its full potential.

The results presented here are going to be used in a case project, which will provide a verification of the usability of this concept in industrial practice. The underlying work is conducted as a co-operation of two projects at the Technical Research Centre of Finland (VTT): a strategic research project focusing on KM and a SPI project for a customer.

We provide background information about the underlying concepts and related terms of KM and reuse in the following and introduce the KM management processes in chapters 2. The process model for KM that we are currently developing will contain an improvement cycle, which is considered important for the success of any related projects. This improvement process is described in chapter 3. In chapter 4, we bring together the methodologies of KM and reuse. Finally, the conclusions of this work are presented in chapter 5.

1.1 Reuse

Reuse can be defined as further use or repeated use of an artefact (or asset). The idea of reuse in software development was first introduced by M. D. McIlroy in 1969 [10] when he proposed a catalogue of software components from which software could be assembled. Since then the software industry has grown exponentially and during this time there have been several reports describing the benefits achieved through reuse, e.g., [11, 12]. However, it has been widely accepted that software reuse offers yet unrealised potential for enhancing software development productivity, and improving the quality and reliability of the end products [8]. From a theoretical point of view, reuse as a development concept offers solutions to many problems in industry. In practice, however, many companies have gained less than expected, and the promises of software reuse remain for the most part unfulfilled [13]. Implementing software reuse has a number of pitfalls, such as making the focus too small, not utilising procedures adopted for reuse, or missing knowledge on how to use the existing possibilities [8]. As a company-wide venture it also carries certain risks. Still, the potential benefits of software reuse have made it one of the most interesting practices that are yet to be fully utilised in software development.

Reuse is commonly divided into two main activities: for reuse and with reuse. For reuse means development of reusable assets and with reuse means building new products by reusing these assets. Fig. 1 illustrates reuse activities and their relation to one another.

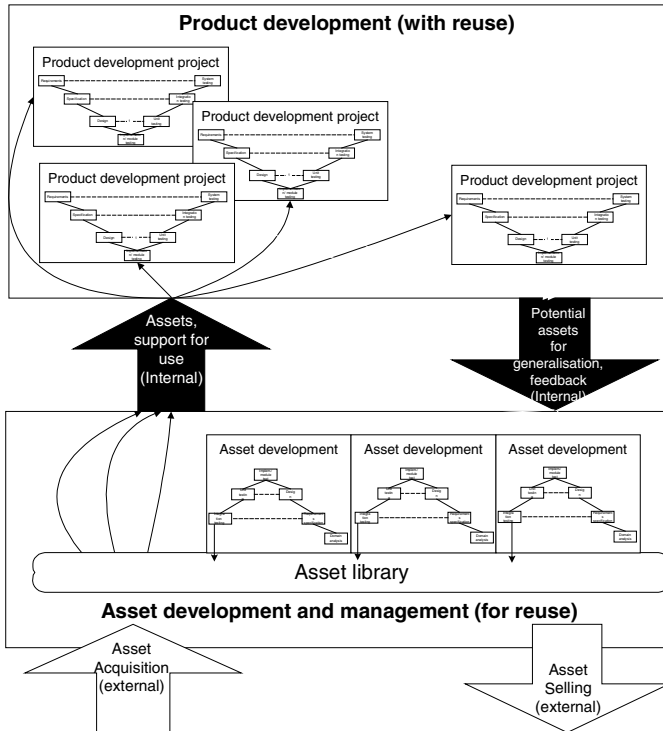


Fig. 1. Reuse activities

The general steps to follow in the development for reuse in our experience are (adapted from [14, 15]):

1. Collect requirements from potential reusers.
2. Analyse requirements and domain.
3. Select and prioritise requirements.
4. Develop and document the solution for selected requirements.
5. Evaluate the asset (testing).
6. Store asset with documentation.
7. Support reusers.

In with reuse, a new product is created by taking applicable assets from the asset base, tailoring them as necessary through pre-planned variation mechanisms, such as parameterisation, adding any new assets that may be necessary, and assembling the

collection. The general steps of the with reuse project in our experience are (adapted from [14, 15]):

1. Find candidate assets for reuse from asset base.
2. Evaluate the candidate assets with respect to requirements.
3. Select assets to be reused.
4. Adapt/modify the assets, if needed.
5. Integrate assets into product.
6. Test.
7. Give feedback to for reuse development.

Reuse also affects supporting processes, that is, the practices needed for supporting the engineering activities. These practices include, e.g.,

- Documentation; assets must be well documented; the greater the asset, the greater the role of documentation. Reusers should give feedback on assets as this information is valuable for the developers and maintainers of assets.
- Quality assurance; quality problems must be identified and dealt with both when developing new assets and when maintaining existing assets.
- Project management; the major difference between managing a project involving reuse and a normal one is the increased number of external stakeholders in the former case, leading to more negotiations and complex trade-offs, as approval is required from all parties (i.e., reusers, developers and management) [15].

1.2 Knowledge and Knowledge Management

There are a number of different definition types for knowledge in literature (see [16] for a summary). The viewpoint common to all of these definitions is that knowledge is something more than just data and information. While data can be defined as a set of facts, and information as a meaningful message, knowledge denotes a combination of human context, experience, values, and expert insights [1]. What this brief definition immediately makes clear is that knowledge is by no means straightforward or simple. Knowledge appears at individual, team, and organisational level, and it exists among people who are affected by the organisation culture, i.e. behavioural norms and practices [17]. A distinction is commonly made between internal tacit knowledge and external explicit knowledge, e.g. as codified into documents, processes etc. [5]. Knowledge is what enables organisations to operate. Or, in other words “...what an organisation and its employees know is at the heart of how the organisations function” [1]. KM, as we use it here, is the activity of managing the creation, sharing and utilisation of knowledge. As knowledge is something complex and mostly internal to human minds, managing the environment is of great importance here. Thus KM deals with building environmental conditions, such as culture, before implementing any concrete sharing processes. The differentiation between knowledge and information is a complex issue [18]; we simply assume that KM covers Information Management (IM), thus also addressing the reversed knowledge hierarchy as introduced by [19].

2 Knowledge Management Process

Several SPI models exist for the field of software engineering (see [20] for an overview) such as the Pr²imer model [21, 22] developed at VTT. However, when approaching the topic KM in general, we realised that no detailed process model, such as the V-Model for software development [23], was available for KM. We concluded to approach KM by starting work on such a process model. In the case described in this paper the developed model is utilised in an industrial environment, which provides verification of its usability.

In this section, a brief introduction is given to the KM Process Model developed at VTT. The processes, 39 of them altogether, are determined in detail. In this introduction, only the main processes of KM are described. Fig. 2 shows the main processes and their interaction between each other.

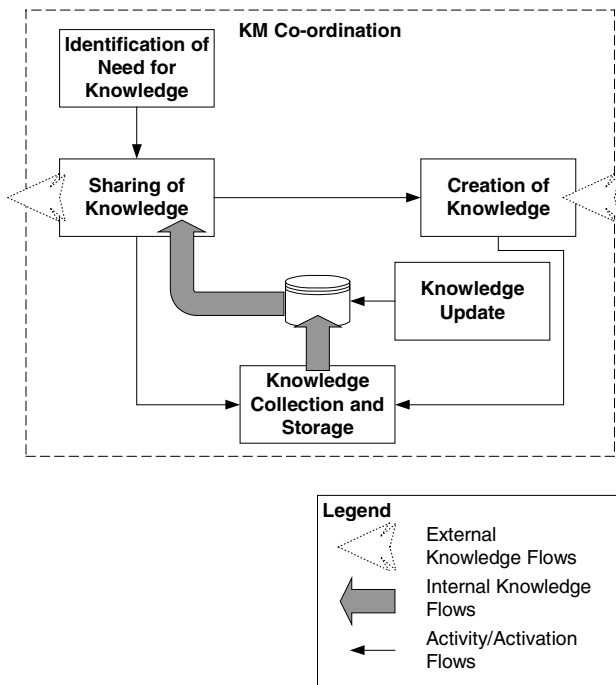


Fig. 2. Main KM Processes

In the following, a short description of each of the main processes is given:

- **KM Co-ordination:** The current state is analysed and a target state is defined as goals and the progression planned in form of processes on basis of the scope of the KM project. The sub-processes constitute an improvement cycle for a continuous revision of the processes.

- **Identification of Need for Knowledge:** Before existing knowledge can be shared, the need for knowledge has to be identified and the requirements for the desired knowledge have to be determined.
- **Sharing of Knowledge:** Whenever knowledge as desired is already present, it should be shared. In addition, knowledge can be shared to the outside world (Knowledge Brokering). Whichever the case, knowledge always has to be set into a proper context. If knowledge is applied to a new context, the result may be new knowledge that needs to be considered for collections and storage.
- **Creation of Knowledge:** If the needed knowledge does not yet exist, it needs to be created. Knowledge creation denotes a combination of knowledge either from inside the organisation, or from the outside (Knowledge Acquisition), or both. External knowledge is connected to Knowledge Creation because of the need to adopt it to the organisational context.
- **Knowledge Collection and Storage:** New knowledge is not only created by demand, but also through inventions and improvements throughout the work. Whenever new knowledge is created, it needs to be identified. The created knowledge can then be evaluated so as to determine whether it will be of further use. If so, it needs to be stored to make sharing possible.
- **Knowledge Update:** Knowledge is context-specific and thus has to change with changing environment. This requires the knowledge stored to be kept up to date. Thus, the changes affecting stored knowledge need to be identified so that they can be analysed towards their impact on the stored knowledge. If needed, appropriate changes need to be undertaken, while outdated knowledge also needs to be discarded.

These knowledge processes are always performed in some way already, which usually is an informal, unconscious and non-technical one. The general goal of a KM project can be formulated as making these processes known to the persons involved, improving them where shortcomings can be identified, and assisting them with technical means where possible and reasonable. Basically, however, KM mainly depends on humans. Therefore, cultural and behavioural aspects are the most important ones, whereas technical means play a secondary role [1]. Nevertheless, technological assistance is regarded as inevitable, once the enabling environment has been established. Especially for address distribution issues, computer-mediated communication and transfer of knowledge are of great importance.

3 KM Process Improvement Model

In this chapter, the KM process improvement cycle is introduced. The main phases of the model are based on Pr²imer process improvement model: Analysis, Definition, Planning, and Effecting. In our process model, this cycle is integrated to the main process KM-Co-ordination. The resulting cycle is shown in Fig. 3.

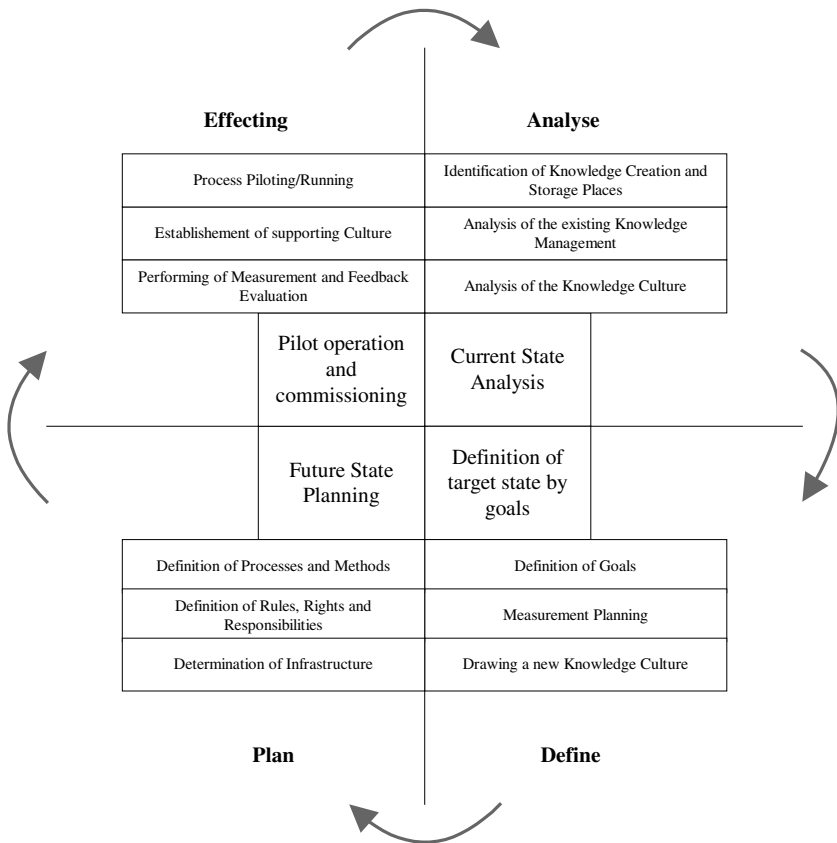


Fig. 3. KM Improvement Cycle

This improvement cycle addresses several issues [1]:

- the need to incrementally approach changes,
- the problem of fast changing technology and working practices [2] and the resulting danger of out-of-date knowledge,
- the poor predictability of costs of KM activities which impedes management decisions, and
- the difficulty of influencing cultural and behavioural practices.

Starting KM with piloting, improving the processes until they work properly, and then widening the area of influence enable a gradually ongoing development instead of a radical change. This allows people to get used to new working practices, and a new cultural environment can be build step-by-step. Also the costs for KM activities can be kept at an acceptable level until success can be seen (and measured) and thus larger

investments become feasible. From the start on, the monitoring and measurement features enable the KM system to keep up with any changes and to avoid outdated knowledge that would endanger KM as a whole [1].

4 KM Improvement in the Software Reuse Process

Based on the findings described above, our approach to an industrial case is introduced in the following. Since the case is in its early phases, this paper will focus on the analysis phase and only an outlook to the other phases of the KM improvement cycle will be given.

In order to improve reuse processes by means of KM process improvement, the reuse and KM process areas that need improvement should be identified first. Thus, the improvement process starts with a current state analysis, in which the reuse processes are assessed, with a special focus on knowledge.

4.1 High Level Analysis of Current State (Analyse)

First, the existing reuse process will be analysed at high level in order to identify knowledge intensive areas in the process for KM improvement. This analysis is done via interviews and workshops. The general steps of the reuse process presented in chapter 1 are addressed. Also, the potential risks in the reuse process are analysed to find out what needs to be addressed in the knowledge management activities.

The results of the general level analysis are evaluated in a workshop together with the reuse process owner, the quality manager and managers from both product and asset development projects. The evaluation results are then used for setting the focus for the detailed analysis of the affected reuse processes, emphasising the knowledge viewpoint. Example focus areas include:

- The knowledge sharing between product development (with reuse) and asset development and management (for reuse). This is especially important if different people in different locations perform these tasks. For example, knowledge concerning future directions for products is needed when evaluating the reuse potential of an asset candidate. When new products are assembled from the defined reuse assets (for reuse), knowledge of the design rationale of these assets will be needed. Also, the so-called NIH factor (Not-Invented-Here) may prevent successful reuse.
- General communication about reuse strategies etc.
- Knowledge sharing between with reuse projects, e.g., how certain assets worked, where they can be used etc.
- Searching for appropriate assets from the asset library.

Improvement actions are performed using incremental approach in order to get some visible benefits early and thus, to keep the people involved motivated and to provide feedback for the management. Number of increments and effort and schedule for the increments depends on increment's focus and are planned separately for each

increment. The focus area for the first cycle is agreed during this high level current state analysis. Each increment consists of the following tasks:

- Detailed analysis of current state (strengths and needs for improvements).
- High level description of target state.
- Drawing up an improvement plan for selected findings including improvement actions, responsibilities, schedule, improvement actions follow-up, metrics, etc.
- Implementation and evaluation of improvement actions.
- Planning the next increment (focus, schedule, and effort).

4.2 Detailed Analysis of Current State (Analyse)

Detailed analysis will be performed via interviews and study of the processes involved. Firstly, the organisational units involved have to be identified and the interviewees determined. In the example focus area of the interface between for and with reuse the interviewed persons will be those project managers and designers of product projects who are using the reusable SW assets in product development.

The interview questions are defined based on the analysis focus and KM process definitions. Each of the KM process areas (KM Co-ordination, Identification of Need for Knowledge, Sharing of Knowledge, Knowledge Collection and Storage, and Knowledge Update) is taken into account when drawing up the interview questions. In the detailed analysis the following questions are addressed, for example:

- What knowledge needs to be managed to address the problem?
- Where is this knowledge located?
- At which points of the reuse processes is this knowledge needed?
- What is hindering the knowledge from being used where it is needed?

The importance of social and cultural aspects for successful KM requires an emphasis on corresponding issues in the organisation, i.e. the environment has to be analysed and possibly influenced towards a supporting culture.

In the example focus area of the interface between for and with reuse the questions to be addressed in interviews include, e.g., the following:

- Are reusable assets searched for during product development? At what point of the development process is this done? How is this search performed? What information is needed in performing the search? Can suitable assets be found with reasonable effort? What problems occur?
- How much adaptation is required for the current needs? Are the assets reusable with reasonable effort? Are the costs predictable for the required adaptation? What information is needed in order to be able to make the needed adaptations? How is adaptation performed? What problems occur?

Here the knowledge sharing process would be of main interest in the interviews. For example, the problem that reuse is not performed, can thus be refined to its roots, e.g., searching is not performed (culture), assets can not be found with reasonable effort (insufficient descriptions or search capabilities), or adaptation is too complicated (improper codification).

In addition to interviews, an analysis of documentation needs to be performed. This enables gaining an understanding of how processes are planned (quality manual) and how processes are supposed to be performed (project reports), thus bringing an additional viewpoint to the interview results.

As a result the strengths and weaknesses in Knowledge Management are identified for the reuse process in the selected focus area.

4.3 Definition of Target State (Define)

In order to be able to plan the improvement actions an ideal or better situation needs to be drawn up for the selected focus area. Therefore, the goals for improvement are defined based on the improvement ideas found during the current state analysis. This will be carried out in co-operation with the reuse project members. The processes designed are regarded as initial and need to be checked against improvements once they are used.

Measurements are then defined to support monitoring the success of improvements by refining the goals to questions and these to metrics according to the GQM-Paradigm [24].

4.4 Improvement Planning (Plan)

The identified improvements are defined as new processes and responsibilities in the improvement plan. In the example focus area of the interface between for and with reuse improvement activities could be the following:

- Introduction of meetings between for reuse and with reuse departments at certain phases of each development project,
- Introduction of general communication channels between the two departments in form of regular meetings, videoconferencing, newsgroups, and newsletters.
- Introduction of a knowledge base designed to address the identified problems to provide easier access to the assets.

The outcome of this stage is highly dependent on the prior stages and of the focus area.

4.5 Implementing and Evaluating the Improvement Actions (Effecting)

The improvements are carried out as a pilot operation according to the improvement plan. The introduction of new techniques and methods is performed step by step, which makes it easier for the people involved in the processes to accept and adapt to the changes. Through this stage-wise approach, acceptance can be easily reached among the people affected, which is elementary for the success of the project. Metrics are used regularly in the processes to enable active tracking of performance and identification of possible problems.

5 Conclusions and Further Work

We have introduced a KM process model and its improvement cycle at a high-level. We have shown our ideas on how our KM model can be applied to reuse processes. Emphasising the analysis phase of the KM improvement cycle we have introduced how we are planning to perform the first KM-based reuse process improvement sequence in a case organisation.

The scenario depicted here, although affecting the basic reuse processes, does not bring any actual changes to them. What we are suggesting in this paper is not a new reuse process model, but a reuse process enhanced by KM processes, which enables addressing problems occurring in reuse. The first improvement cycle we have introduced is merely intended as the first piloting of these means. Any shortcomings identified during the pilot operation can be addressed in further improvement cycles. In addition, an extended focus can be applied to widen the area of impact. Each further improvement step concerning content or scope will take a cycle of its own in the KM improvement process. This ensures that the most important aspects are taken into account and addressed accordingly.

The methodology of KM-based reuse process improvement allows piloting and improving the reuse process, and once changes are working spreading them for wider use. The general aim being a process of continuous improvement, we are planning to perform the first one or two cycles together with the customer, assuming that the following cycles will be performed without us. However, we will be monitoring this activity so as to be able to verify the applicability of the developed KM process model and to address any shortcomings on the methodology identified during this work.

References

- [1] Davenport, T.H. and Prusak, L. Working Knowledge - How Organizations Manage What They Know. Boston, Massachusetts: Harvard Business School Press; 1998. ISBN 0-87584-655-6.
- [2] Carneiro, A. How does knowledge management influence innovation and competitiveness? in: *Journal of Knowledge Management*; Vol. **4** (2). pp. 87-98.
- [3] O'Dell, C. and Grayson, C.J., Jr. If Only We Knew What We Know: The Transfer of Internal Knowledge and Best Practice. New York: The Free Press; 1998. 238 pp. ISBN 0-684-84474-5.
- [4] Marler, K. Rapid Emerging Knowledge Deployment in: *Crosstalk: The Journal of Defense Software Engineering*; Vol. **12** (11). pp. 14-16.
- [5] Nonaka, I. and Takeuchi, H. The Knowledge-Creating Company. Oxford, New York: Oxford University Press; 1995. 284 pp. ISBN 0-19-509269-4.
- [6] El Emam, K.; Drouin, J.-N. and Melo, W. SPICE: The Theory and Practice of Software Process Improvement and Capability Determination. Los Alamitos: IEEE Computer Society; 1998. 486 pp. ISBN 0-8186-7798-8.
- [7] Fitzgerald, B. and O'Kane, T. A longitudinal study of critical success factors for software process improvement in: *IEEE Software*; Vol. **16** (3). pp. 37-45.
- [8] Jacobson, I.; Griss, M. and Jonsson, P. Software Reuse: Architecture and Organization for Business Success. New York: ACM Press; 1997. 497 pp. ISBN 0-201-92476-5.

- [9] Hilburn, T.B.; Hirmanpour, I.; Khajenoori, S.; Turner, R. and Qasem, A. A Software Engineering Body of Knowledge. Carnegie Mellon University (1999). Online: <http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr004.pdf>.
- [10] McIlroy, M.D. "Mass produced" Software Components in: P. Naur and B. Randell, Editors: *Proceedings of the 1968 NATO Conference on Software Engineering*. Brussels: NATO Scientific Affairs Division; 1969. pp. 138-155.
- [11] Lanergan, R.G. and Poynton, B.A. Reusable Code - The Application Development Technique of the Future in: *Proceedings of the SHARE/GUIDE/IBM Applications Development Symposium*. Monterey, California: IBM; 1979. pp. 127-136.
- [12] Tajima, D. and Matsubara, T. Inside the Japanese Software Industry in: *Computer*; Vol. **17** (3). pp. 34-41,43.
- [13] Mili, A.; Yacoub, S.; Addy, E. and Mili, H. Toward an Engineering Discipline of Software Reuse in: *IEEE Software*; Vol. **16** (5). pp. 22-31.
- [14] Lim, W.C. Managing Software Reuse, A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components Prentice Hall; 1998.
- [15] Karlsson, E.-A. Software Reuse: A Holistic Approach. Chichester: John Wiley & Sons; 1995. 510 pp. ISBN 0-471-95489-6 / 0-471-95819-0.
- [16] Spiegler, I. Knowledge Management: A New Idea or a Recycled Concept? in: *Communications of the Association for Information Systems*; Vol. **3** (
- [17] De Long, D. Building the Knowledge-Based Organization: How Culture Drives Knowledge Behaviors. Working Paper, Cap Gemini Ernst & Young Center for Business Innovation, 1997. Online: <http://www.businessinnovation.ey.com/mko/pdf/wculture.pdf> (accessed: 2001-02-19).
- [18] Leonard, D. and Sensiper, S. The Role of Tacit Knowledge in Group Innovation in: *California Management Review*; Vol. **40** (3). pp. 112-132.
- [19] Tuomi, I. Data is more than knowledge: Implications of the reversed knowledge hierarchy for knowledge management and organizational memory in: *Proceedings of the 1999 32nd Annual Hawaii International Conference on System Sciences*. Los Alamitos, California: IEEE Computer Society; 1999. pp. 45-.
- [20] Rada, R. and Craparo, J. Standardizing Software Projects in: *Communications of the Association for Computing (ACM)*; Vol. **43** (12). pp. 21-25.
- [21] Pr2imer. Webpage, Technical Research Centre of Finland (VTT), 1999. Online: http://www.vtt.fi/ele/research/soh/products/primer_etusivu.htm (accessed: 2001-04-18).
- [22] Karjalainen, J.; Mäkäriäinen, M.; Komi-Sirviö, S. and Seppänen, V. Practical process improvement for embedded real-time software in: *Quality Engineering*; Vol. **8** (4).
- [23] The V-Model. Webpage, German National Research Centre for Information Technology (GMD), 1996. Online: <http://www.scope.gmd.de/vmodel/en/> (accessed: 2001-03-19).
- [24] van Solingen, R. and Berghout, E. The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. London: McGraw-Hill; 1999. 199 pp. ISBN 0-07-709553-7.

Processes and Knowledge Management: A Symbiosis

Christof Nagel

T-Systems
T-Nova Deutsche Telekom Innovationsgesellschaft mbH
Development Center South West
PO Box 100833
66008 Saarbrücken
Germany

Christof.Nagel@t-systems.de

Abstract. In the Development Center South West the business and engineering processes and the knowledge management system build a symbiosis. The knowledge management system is embedded in the business process model, which has been established for several years. The processes provide the knowledge management with input and support the use of it. The system manages information objects, which contain the description of knowledge, “non practices” and “best practices” of the knowledge management and which have a structure, conceptual based on the quality improvement paradigm. Beside the usual ones (queries, ...) a new kind of accessing information objects is given by the processes, whereas the processes and their activities are attributed with links to the objects. A systematic approach for improving processes is a result of the symbiosis. This article describes the results we have got one year after initializing the symbiosis.

Introduction

The T-Nova Development Center South-West is a software producing unit in the area of telecommunication systems. It builds software systems for the Deutsche Telekom AG and for other clients from the industrial and from the government sector. It has introduced a knowledge management system based on the business process and in order to support the processes. The knowledge management system does both, it enhances the models of the processes and improves the application of them. The last conference on professional knowledge management ([6],[5]) has shown now that knowledge management and processes can not be divided.

Although all processes of the business process model are covered the knowledge management system focuses on the support of the software engineering and consulting processes. The knowledge of the Development Center based on experiences should be documented and represented. As the Development Center has a

well defined system of business and engineering processes several advantages were used during introduction and development of the knowledge management. But also after introduction the knowledge management is supported by the processes. The sources of getting input for the knowledge base and the application of the knowledge are also part of processes. One of our objectives two years ago was "Let each member in the Development Center know what the "best practices" and experiences of the other colleagues are." This objective was the starting point for the development of our concept for knowledge management. During the development we had to find answers on the questions:

- What is the input we like to have in the knowledge management system and how do we get the input by using the processes? Chapter "Process of Knowledge Management"
- What are the elements/objects of our knowledge management system and how is the data base organized? Chapter "Development of the Basic Concept"
- How should our users get access to the contents of the knowledge management system? And how can the knowledge data base be connected to processes? Chapter "Development of the Basic Concept"

In the following of this article we give an answer to these questions in our point of view. Based on the answers we have build up a system and an approach which can be characterized as follows:

- Nearly all what we do in the knowledge management is supported, based or guided by the processes. The processes provide us with input for the knowledge management system and they assure that we get information about the use of the packages from the knowledge management system.
- The view on the data base of the knowledge management system is not given by an ontology or structured by a knowledge tree as discussed and proposed in literature. We have not seen an advantage in developing a knowledge tree. One view to look on the knowledge data base are processes. The processes are an ontology approach, whereas this ontology is well known in the Development Center. The knowledge is derived from the processes namely the software engineering processes. The knowledge is also linked to the processes and these ones are the knowledge trees used in this approach.
- The objects of the knowledge management system are packages having a structure influenced by the cycle of the quality improvement paradigm.

Development of the Basic Concept

Several years ago we have studied the basic concepts of the quality improvement paradigm [4] and of the experience factory [3]. The quality improvement paradigm is based on a cycle, an improvement cycle, consisting of the steps:

- characterize and understand problems
- define goals
- select suitable process methods, techniques or tools
- perform the methods or techniques or apply the tools

- analyze the results
- package the experiences and put it in a data base

First experiments with the quality improvement paradigm have shown, that the approach does not fit exactly for the Development Center, as it has defined and uses processes covering each business. That means we have processes for example for management of customer relations, personal support, production processes for software engineering and consulting, An important role in this context has the improvement process. A part of this process are assessments, based on Bootstrap-method, being applied to units (projects, departments, teams) of the enterprise. By the assessment the definition, the right customization and the usage of the processes especially the software production process are controlled.

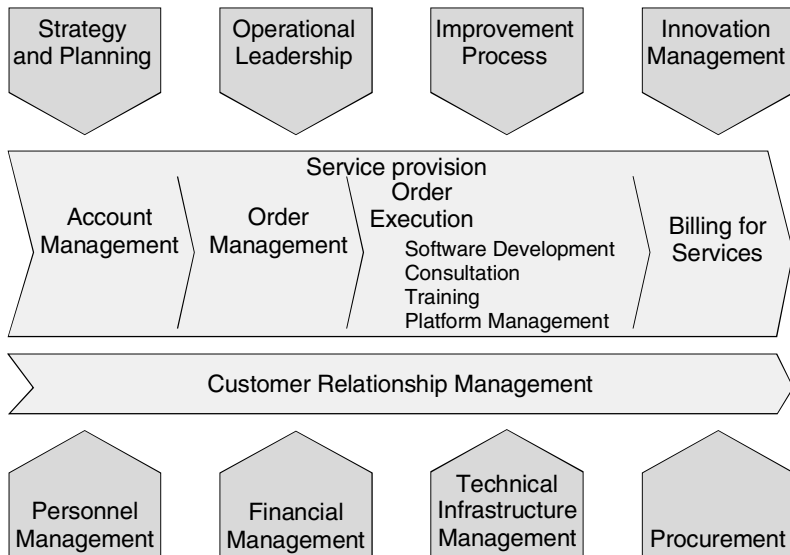


Fig. 1. The business process modell of Development Center South West

In the assessments metrics and characteristic numbers are analyzed in order to determine whether they fit according to the goals and objectives of the unit. In the case they do not fit or in the case of other weaknesses of the unit we define improvements concerning new metrics, management or engineering practices. The unit has to work out an improvement plan and to perform the improvements. The results of the improvements are again checked by assessments. Taking the points described here into account, then the different parts of the improvement process and of our assessment procedures cover already a number of steps from cycle of the quality improvement paradigm.

Another part of the cycle is covered by the process for the derivation and definition of goals and objectives. Therefore there was no need to introduce the paradigm formally in departments or teams. In counterpart the introduction of the paradigm would instantiate a second and a little bit different improvement process in the

enterprise. As the processes contain already the main parts of the quality improvement paradigm we have decided to develop a knowledge management system which is based on the ideas of the quality improvement paradigm and collaborates with the processes.

The Information Objects of the Knowledge Management System

The information objects in this system are structured. A number of products bought under the name knowledge management systems have a lot of function in order to handle information objects being unstructured but have often not sufficient functions to handle structured objects. But a structure in the objects is necessary, as the users of the systems, that means the readers of the objects, expect to find the information they search fast and as quite as possible easy. The users like to have the same look and feel for each information object. The information object in our case are called experience packages, as they document mostly experiences made by units or members of the enterprise. Usually each experience has its own package. One package should not contain two or more experiences. The structure of the packages does not allow this and the handling of packages with more than one experience is difficult especially the searching. In the structure of a package parts can be detected, which correspond to certain steps of the cycle from quality improvement paradigm.

- title: Each package has a title, which is built from the main message of the package.
- abstract: The abstract gives a short summary of the package in one or two sentences. This should help the reader to decide whether the package is interesting or not.
- problem or state: In the most cases of an experience the people have had a problem or state of their work or their project. The problem had to be solved or state had to be improved. The original problem or state is described in this part.
- method, procedure: In this part of a package the methods or procedures to solve a problem or to improve a state are described.
- results: By introduction of a new method or procedure a result is intended. But is the result, got after performing the method or procedure, the one which was intended? Very often we have differences between the desired and the real result. The real result and the deviations are described here.
- downloads: During the performing of new methods or procedures useful programs, utilities, macros or templates were created and used. In the experience packages are links set to these tools, so that readers can download them.
- comments: Each reader of a package has the opportunity to write a comment to a package. The discussion of new methods or procedures is supported by this way.
- characterizing, identification: Some other parts are contained in a package in order to characterize, sort the package and to identify the author. The author has also the role of a contact person in cases where a reader has questions to a package.
- classification process: A special part contains the information for attaching the package to the processes. By writing the corresponding ID, it is here described to which processes, subprocesses or activities the package has to be attached to. Usually more than one ID is given in this part.

Title	Collection of topics for status meetings
Abstract	Systematical preparation of status meetings
Problem	The results of status meetings are unsatisfactorily: Topics are forgotten or are not well prepared.
Method/Procedure	A template with the agenda is provided for team members. It is put in a directory accessible for each member. Everyone of the team write his topics in the agenda for the next meeting. It is an obligation for the project manager to prepare on the topics and to discuss them in the next meeting.
Results	The template is mostly used by the project and quality manager. The quality of the meetings was improved significantly.
Downloads	You can find the template under this link .
Contact Person	Gerd Schwantke
Comments	none
Classification (Process)	6c1QD06
ID	EP_1055
Datum	12.07.2000

Fig. 2. Example of an experience package

Access to Knowledge Management System

How do the users of the system get access to experience packages? A main difference between concepts for knowledge management systems described here and in the literature or discussed in workshops lies in the so called structure tree or knowledge tree. Very often it is proposed to invest a lot of work in the definition of an ontology and in the creation of a well defined knowledge tree [1]. Before a lot of work is spent, it should be analyzed what have to be done in order to introduce a knowledge tree and what is the probable return of invest. What are the purposes for the construction and maintenance of tree, if the knowledge management system offers one?

- Informations and contents of the packages were extracted and generalized to topics. The topics are mapped on a tree oriented structure, whereas the categories represents the levels in the tree. By that way a selection of the content from the packages is copied into a tree oriented structure.
- The definition of the knowledge tree has to be reviewed or and maintained at least yearly.

Collecting these facts and some more described in [7], comparing the effort of definition, construction and maintenance with the advantages of the tree, we got the opinion that the return of invest of the definition of an new ontology and the creation a knowledge tree is only minimal. We prefer to use an natural and well understand ontology. This ontology are the processes of the enterprise. Our knowledge management system offers the users three possibilities to access to experience packages:

- Navigation to the experience packages via our processes

- Different lists of packages for example: list of new packages or list of top ten packages
- Find the packages by using a search engine

The last both ones are not a topic of this paper they are described in [7].

Navigation to Experience Packages

The T-Nova Development Center South- West has as already mentioned before a well defined system of business and engineering processes. Each business and engineering activity is described in or covered by a process. The processes are described in the following manner: In level 0 the whole business process model is defined. The level is shown in Fig. 1. The next level contains the processes itself with their refinement in sub processes. In level 2 the processes are described in detail with activities, roles and important input or output documents of the activities. The process descriptions are not any documents which are only written for getting an ISO-certificate or something else. The process descriptions are used for the work. Especially the description of the software process are taken in order to generate project plans, to support effort estimations, to get hints for what is to do in the specific subprocesses or something else. A base for that is a customizing of the standard software processes in order to get a project specific process.

The idea, which was realized, was the use of these processes in order to represent the experiences of the enterprise. If the definition of the business process model is complete and detailed enough, the great majority of experiences the people in an enterprise made can be attached to processes and their activities. The experiences described by the packages are attached to the activities and subprocesses. Our approach has several advantages:

- By coupling of activities of processes and experience packages we have on one side the standard description of the activities how they have to be performed and on the other side the experiences. The experiences explain how the activity can be done or offers a solution (a pattern, a tool, a macro, a template...) or describes probable problems occurred during the performance of an activity. The members of the enterprise do not get only the standards of the processes and their activities they have to perform but they get also examples or solutions how they can do it.
- The processes cover the business and nearly each activity of the business. By attaching experiences to the processes points of similar situations can be detected which different teams/departments of the enterprise have in performing a process. If it can be considered that certain activities or subprocesses have got a greater number of experience packages attached, then these packages have to be analyzed. Do the packages express or describe similar performances of process activities, in that case the packages should be used to improve the processes or a part of them. This feature will be described later in detail.
- The processes are defined and introduced 5 years ago. They are known and used in projects and departments of the enterprise. The degree of knowledge about and the use of the processes are monitored by audits and assessments. If someone have a question he or she should identify which activity or subprocess is involved and can look whether there are experiences attached.

In a certain way the processes and their attachments of experience packages can be understand as an approach of an ontology or a knowledge tree. But this tree was already defined and the processes are well known to the members of the enterprise who perform the processes. The tree is a very natural one, being build from the structure and from the business the enterprise has. The technical realization looks as follows. The processes are represented in an intranet. Each subprocess and each activity having at least one experience package attached has got an additional information "KM(*)", which express how many packages are attached to (see Fig. 3).

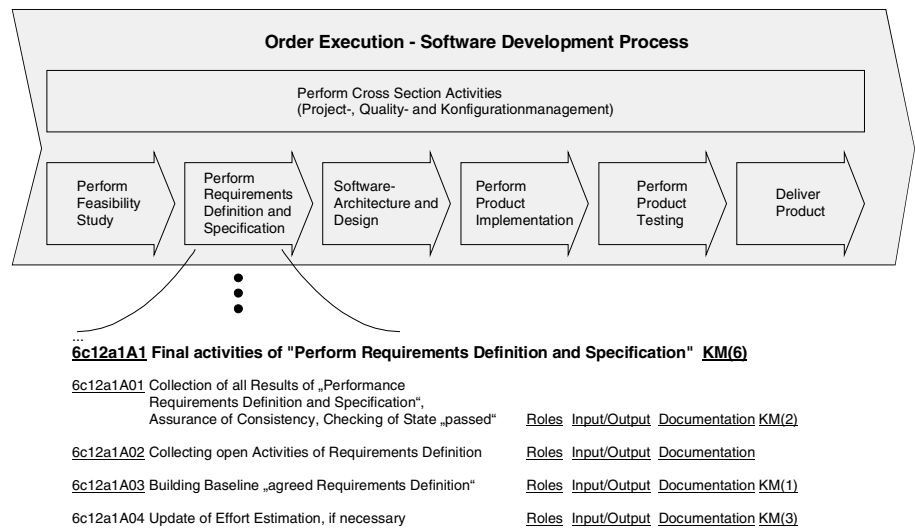


Fig. 3. Software development process attributed with links to the knowlege management system

The number is also an link to intranet server of the knowledge management system. By traversing the link a list of links to the packages itself is shown. Each item in the list consists of the title and a short abstract. The list is sorted by date. The layout of the list is the one shown in Fig. 4. Based on the title and on the abstract the reader can decide whether or not to navigate to the package

<u>"Collection of Topics for Status Meetings"</u>	
Abstract	"Systematical Preparation of Status Meetings"
<u>"Integrated View on all projected Processes"</u>	
Abstract	"In the Sense of EFQM and TQM there must be an integrated View on all Project Processes."
<u>"Progress Control in Projects"</u>	
Abstract	"Progress Control in Projects with MS-Project and Outlook"
...	

Fig. 4. List of published packages

Process of Knowledge Management in T-Nova Development Center South-West

The process of knowledge management consists shortly described of the following steps:

- At first we have different sources to get input. The sources deliver inputs in different qualities and different details to an experience.
- The input is analyzed. It is determined whether the input is interesting for knowledge management. Missing details are demanded from the people who have given the input. A package is formed from all of the input to an experience.
- The packages are collected and embedded in the data base. The attachments to the processes are performed. The lists of packages are constructed.
- The packages are taken for consulting the departments and project teams.

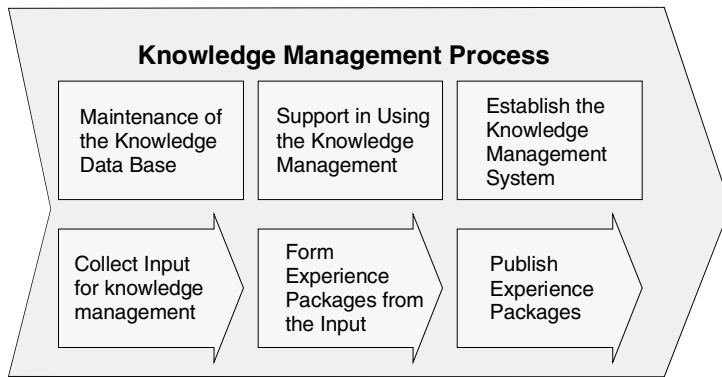


Fig. 5. Process of knowledge management

Sources for Input to the Knowledge Management

Different sources are used for the knowledge management. One of the sources has their origin in spontaneous inputs given by members or teams of the enterprise. Supported by a tool the experiences can be written in a very easy manner. The tool serves as input source for the knowledge management as well as one source for the improvement process. The layout of the input form of the tool is a result of discussions with the potential users. The main points in the discussions were:

- Keep the tool easy. The tool has to be self-describing, so that the use is easy.
- Ask only for information being real necessary. For example avoid to ask for classifying details.
- Give a fast feedback what will happen with the input.

The tool is accepted in the enterprise. The number of inputs got by the tool meets the expectations. Not only experiences are accepted as input for the knowledge management, but also knowledges or know how. This source of experiences is based

on a bottom up principle. The people are free to give an input or not. There is no process or standard or something else which forces the people. It is important to have this source where the using is absolute free. Up to now we have considered that a lot of the input from this source is put in the knowledge data base. But the resulting experience packages have an effect on process improvement only in a few cases. The reason is, that this source is used to communicate know-how, which has a more technical character.

Another source of inputs is embedded in the software engineering processes. In the engineering processes of the business process model a special review type, the project review, is defined. In a project review the experiences of a project team are considered. The team is asked what are the experiences which should be documented in the knowledge management system that other colleagues can read and use them. After attaching these experience to the software engineering process, they show how the standard of the process can be performed, which alternative and which solutions or templates do already exist.

The experiences are documented at first in a review report. In this report the experiences have a structure being very similar to the packages in the knowledge management data base. The experiences documented in a project review report can be classified as "best practices", "non practices" and informations. The last ones are not experiences they are often knowledges or recommendations. But these informations are also worth to be taken in the data base. It was surprising that the teams want also document the "non practices". "Non practices" have their reason often in a mistake of someone. The packages resulting from these "non practices" are very important and leads to new improvements of the processes, as it is described later in this article. Based on the fact that the teams document "non practices" it can be followed that they want publish their experiences by the knowledge management and that the project reviews are accepted. The acceptance is necessary in order to get qualitatively high project review reports. The documentation of "non practices" has a further effect. These practices demonstrates or gives hints where the processes can be improved (see section 4).

A third source of experiences are the assessments, which are based on the Bootstrap Method and focus on the processes needed to build software systems. During assessments being applied to software projects the assessors detect a method or a solution by which the software engineering process is very well performed. These method or solution is considered as a "best practice", can serve as a good example how the standards of the process can be performed and should therefore be represented in the data base of the knowledge management system and should be linked to the process in a manner already described. This practice is documented in the result report of the assessment. The knowledge engineer gets a hint to the "best practice" and decides whether or not the practice can be accepted for the knowledge management.

In a lot of cases the informations to the "best practice" does not allow the decision so that it is necessary to ask for more details to the "best practice". Together with all informations an experience package is formed and put in the data base. By this source of experiences we concentrate only on "best practices" as the "non practices" should only be published via knowledge management by the assessed team or departements itself and if they want that. As only the "best practices" are published this source is

well accepted in the enterprise. Using the Bootstrap-Assessments in that way has two advantages.

The project, being assessed, gets recommendations how the project processes can be improved. The "best practices" can serve as good examples for other projects and can be used to improve the standard processes. The last two sources have a top down character. Given by the engineering process and the improvement process the teams and departments have to perform project reviews and assessments. This approach has advantages:

- The approach to get input has a systematic part and is not only based on the spontaneous input of the members of the enterprise.
- Detection of potentials for optimizing or improving the project specific processes, but by documentation through the knowledge management it becomes evident that the standard process is the one which should be optimized or improved. Especially the project reviews have shown after the attachment of experience packages to the processes, that the optimization and improvements of the processes are possible which nobody has known before.
- It is no problem to get input as great number of reports from assessments and project reviews are written yearly.

Forming Experience Packages, and Linking to Processes and Publishing

The input for the data base has to be transformed in the structure of an experience package. Often the input does not have any structure. The purpose of the knowledge engineer is to form the experience package from the input and to publish the package. The purpose is detailed in:

- Transformation of the input into the structure of an experience package.
- Insertion of a title and of an abstract in the package.
- The complete business process model is analyzed in order to find out to which process or to which activities of the processes the experience package can be linked. The processes and their activities have identity numbers (e.g. 6c1QD06). The identity numbers of the processes and their activities are inserted in the package.
- In the next step the packages are released for the intranet servers of the knowledge management system and of the business process model.
- The last step consists in asking the process managers for a comment to the package. The comment will be also published. The packages are not censored by the process managers. Of course, each member of the enterprise can write a comment to a package too. Comments to comments are desired.

Support in Using the Knowledge Management

Beside other successful approaches in our concept a strong support is guaranteed by our assessors. They have not only the role of giving hints to "best practices" as described in section "Sources for Input to the Knowledge Management". They have also the purpose to consult with packages. The consulting consists in proposing

packages and their application in order to help the teams or departments to improve their processes and hereby themselves. The assessors get always the newest information to the knowledge data base. During the setting up of a new project the knowledge engineer analyzes together with the project managers the main characteristics of this project. A data base research is performed to find packages being helpful and interesting for the project. The knowledge engineer consults the project manager in the application of the packages.

This procedure has some advantages. The new project is supported with the newest experiences from the data base. The support is optimized by the knowledge engineer as he knows the content of the data base very well. Mistakes made in earlier projects can be avoided. But the knowledge management is also supported as the contents of the data base are used. During the consulting the knowledge engineer learns which contents of the data base are demanded and which not. This will have an influence on the topics discussed during the project reviews. In our enterprise the knowledge engineers, the consultants and the assessors come from the same team. This constellation supports also the use of the knowledge management system, as the same team

- is responsible for the process quality in the project teams or departments,
- consults the teams in applying the processes, standards and engineering methods and
- is responsible for the content of the knowledge data base.

Having all these purposes in one team help to keep the quality of the data base high and to detect how and in which teams the content of the data base could be applied.

The Symbiosis

In the previous sections some of the detailed collaborations between the different paradigms (Bootstrap, processes, knowledge management, ...) were described. But our concept contains more than collaborations it is a symbiosis of the different paradigms. Each of the paradigm needs at least one of the other for an optimal effect. Since we have get the symbiosis by establishing the knowledge management the process improvement itself has got a new and better quality.

Already a few month after introduction of the knowledge mangement system some results could be considered. After a while of putting experience packages in the data base and linking them to the processes it was determined that a greater number of packages have the same topic. This determination is done by an analysis of the processes. Each activity of a process, to which a greater number of packages is attached to, is analyzed in detail together with the packages. The question is: Do the packages or some of them describe similar situations? If it is so, then there is an accumulation point in the process at this activity. In the next step the packages have to be analyzed and improvements or changes to the process have to be performed.

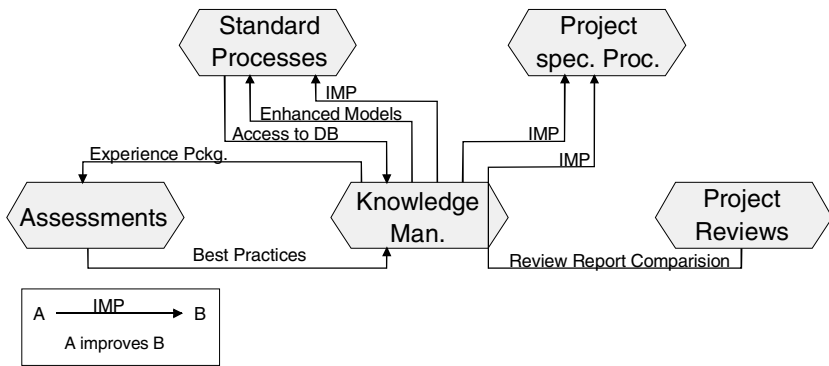


Fig. 6. The symbiosis

A further improvement effect was got by the analysis and comparison of the project review reports. These reports show the problems of the project teams. They show how the teams act in similar situations and also the differences between the teams. Based on the reports the knowledge engineer is able to consult the teams in order to solve the problems by giving examples from the practice of other projects. Problems which have several teams in a departement are discussed with the managers of the department.

The knowledge management gets from the assessments best practices and from the project reviews both best and non practices. For using in project assessments the knowledge management provides the assessors with the informations from knowledge data base. The informations contain for example experience packages with best practices or from the knowledge data base new analysis results, describing new conclusions concerning the performance of the processes. The project itself use the knowledge data base via the standard processes or via the data base entry in the intranet to look for experience packages being interesting for the project or the project staff. But additionally there is also a systematic approach where a knowledge engineer analyzes the characteristics of a new project, looks for the most important experience packages based on the characteristics and consults the new project with these packages.

The models of the processes are enhanced by a new view. Each process, each subprocess and each activity has an link to the knowledge management and to experience packages. The process definition and the standards define what to do is in performing the process. The experience packages can explain how it can be performed. And sometimes there are even solutions for performing the process or an activity of it.

Results

The Content of the Knowledge Data Base

Since one year working with the approach we have collected input for the experience packages. For that the three sources for getting input described in section "Sources for Input to the Knowledge Management" are used. 120 packages are stored in the knowledge data base and the number is growing continuously. The quality and the size (measured in words of description) are very differently. The criterions for the quality concentrate in the question: "Does the experience package have a potential use or advantage for someone or for the processes?" We intended to encourage the people of the enterprise to give us input via one of the three sources. Up to now we think it was the right way as:

- The data base was filled with a sufficient number of packages in an acceptable quality.
- The people in the enterprise learn that it is worth to work with the knowledge management system.
- A lot of packages show in practice how to perform the processes, their activities and their standards.
- The quality of the packages is quite good.

The topics treated in the packages were not limited to any area of interest. Although that there was no special area of interest most of the packages can be linked directly to activities of the business process model, where the majority of the packages are linked to the software engineering process. This depends both from the sources for getting input used in this approach and that the business process model covers the whole business of the enterprise. Most of the packages describe problems and solutions for improving the performance of processes. Packages resulting from spontaneous inputs describe often new ideas. It was an advantage that the input sources, i.e. assessments and project reviews are embedded in the process model as:

- The writing of input in project reviews is integrated in the usual work, which is documented in the software process since several years.
- There is only a small additional effort to give input for the knowledge management. The input got from the assessments is even spin-off.

Examples for Process Improvement

A few of the process improvements are represented here.

After the first consideration of the processes attributed with experience packages it can be seen that two activities of the software process have a greater number of packages. The packages concern the subprocess test of software and especially the test case documentation and test automation. By using the experiences from the packages the application of a tool for test automation was optimized. Mistakes made before and documented in the experience package were avoided. Further

improvements are planned by defining of recommendations for standard test tools for test documentation and automation.

A best practice were used to improve the project management. Some projects have read a corresponding experience package and have tested the utility contained in the package. Now they use the utility in order to control the project teams by smaller orders to member of the team and they monitor the efforts more effectively.

Conclusions

In this article we have shown the symbiosis of processes and of a knowledge management system defined and established at T-Nova Development Center South-West. The selected approach is a pragmatic and a successful one. It is pragmatic as existing concepts for knowledge management are adapted and mapped to our processes in a practical manner. It is a successful one as the knowledge management is accepted by the member of the enterprise. One reason for the acceptance are the processes, which have allowed to introduce and to run a knowledge management system with only a small additional effort. The introduction and the establishment of the system would have been more difficult without the business process model. The processes itself profit as the improvement is more systematic since the introduction of knowledge management. Our conclusion is knowledge management can not be run without processes and processes need the knowledge management for improvements.

- [1] Probst G., Raub S. and Romhardt K.: Wissen managen - Wie Unternehmen ihre wertvollste Ressource optimal nutzen. Frankfurter Allgemeine - Gabler (1997)
- [2] Gensch P.: Inhaltliche Gestaltung wissensbasierter Datenbanken. Seminar: Ideen- und Projektdatenbanken, Management Circle (1999)
- [3] Rombach D., Bomarius F. and Birk A.: (internal) Workshop Experience Factory (1997)
- [4] Basili V., Green S.: Software Process Evolution at the SEL. IEEE Software (1994) 58-66
- [5] Abecker A., Maus H., Bernardi A.: Softwareunterstützung für das geschäfts-prozess-orientierte Wissensmanagement, Professionelles Wissensmanagement - Erfahrungen und Visionen, Shaker Verlag (2001)
- [6] Schnurr H.-P., Staab S., Studer R., Stumme G., Sure Y.: Conference on Professionelles Wissensmanagement - Erfahrungen und Visionen, Shaker Verlag (2001)
- [7] Nagel C.: Knowledge Management: A pragmatic process based approach, in Software Quality, Springer Verlag (2001)

Augmenting Experience Reports with Lightweight Postmortem Reviews

Torgeir Dingsøyr¹, Nils Brede Moe², and Øystein Nytrø^{1,2}

¹Department of Computer and Information Science,
Norwegian University of Science and Technology
Currently at: Fraunhofer Institute for Experimental Software Engineering,
Sauerwiesen 6, 67661 Kaiserslautern, Germany
dingsoyr@idi.ntnu.no

²SINTEF Telecom and Informatics,
7491 Trondheim, Norway
(Nils.B.Moe|Oystein.Nytro)@informatics.sintef.no

Abstract. Many small and medium-sized companies that develop software experience the same problems repeatedly, and have few systems in place to learn from their own mistakes as well as their own successes. Here, we propose a lightweight method to collect experience from completed software projects, and compare the results of this method to more widely applied experience reports. We find that the new method captures more information about core processes related to software development in contrast to experience reports that focus more on management processes.

1 Introduction

Many small and medium-sized companies that develop software seem to experience the same problems in several projects, like using more effort than originally planned, for example in the test phase. To reduce the impact of such problems, project managers would often like to know what preventive actions other projects in a similar situation has taken, and what the results of these actions has been. Other projects might experience technical problems with, say a compiler, that they know have appeared in the company before, but it is difficult to find which people were involved in solving the problem then. Very few companies have systems that will capture and share this type of information.

Another characteristic of small and medium-sized companies that develop software is that they are usually under strict time pressure. They do not have the time to invest in prevention of possible future problems in the project. Usually, projects are also pretty small, involving typically between 5-10 people, which also means that they cannot use a lot of resources on this.

Here we suggest a *lightweight* method to capture experiences from completed software projects, that is suitable for companies focusing on learning from their own experience. The reasons can be either to improve the quality of their products, be

more efficient, or to make the company a stimulating place to work. This is often referred to as *knowledge management* [1], and involves collecting, refining, packaging and distributing knowledge within a company.

Within software engineering, to focus on knowledge management has often been called to “set up an experience factory” [2]; an organizational framework for capturing and reusing experience from accomplished software projects. The idea here is to allocate resources for managing company-internal knowledge (the “experience factory”). –This unit should collect experience from ongoing and completed projects and make this available for others. It does not have to be a separate part of an organization, but some people should have a responsibility for it.

An interesting question in knowledge management in general, and particularly within software engineering, is how to collect, “harvest”, or “make explicit” experience from projects so that they can be usable for others. What efficient methods exist, that do not require a lot of effort to make requisite knowledge available? In this paper, we are interested in looking at different lightweight approaches to capturing experience from projects (indicated with the bold arrow in Fig. 1), and in particular projects that are completed. This can be a way to make *tacit* knowledge [3] present in software development project *explicit*, and to store it in a knowledge repository or “experience base” to make it available as support for future projects. It can also be a source of data for improvement activities, which will be of interest to the management or “sponsoring organization”. By “experience package engineering” in the figure, we mean to collect and make available experience on different topics in a way so that they are easy to use.

We will look at two different methods for capturing experience:

- *Writing Experience Reports*, reports written by project managers to sum up experience from the projects (single-author documents)
- *Conducting Postmortem Reviews*, a group process to investigate problems and successes in a project. We have developed a particular lightweight method, which does not require much effort.

To see differences between the methods, we have examined resulting reports from two software companies. Now, we first discuss the nature of experience, before saying what we mean by Experience Reports, Postmortem Reviews, and in particular *lightweight Postmortem Reviews*. We then give some context by describing the research project where this work was performed, as well as the companies where we collected the experience. Next, we limit the scope of this paper, and go on to present the research method applied here in section 2, results from each method in section 3, discuss them in section 4 and conclude in section 5.

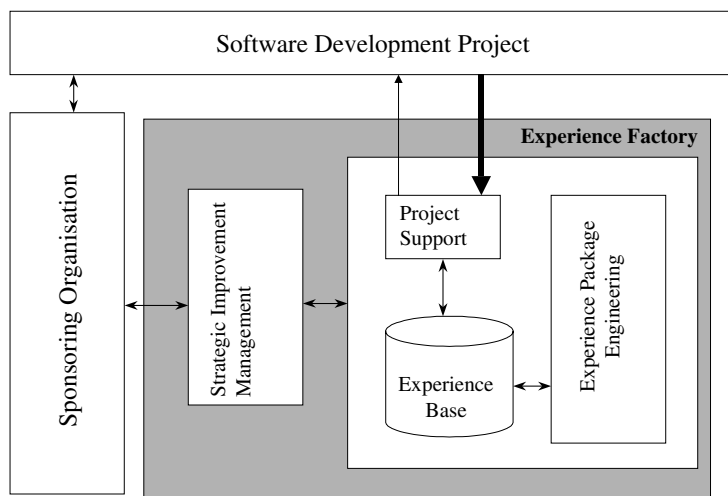


Fig. 1. Experience Factory organization, with experience capture from projects shown in bold. (Taken from the Perfect Project [4]).

1.1 What Is Experience, and What Forms Does It Take?

Let us start by defining a related, but more broad term, “knowledge”. Here, we will use “knowledge” in a quite wide sense, meaning information that is “operational”, that is, usable in some situation.

Experience in a strict sense is something that resides in humans, and that is not possible to transfer to others, because you have to experience it yourself to call it an experience. We will use the word in a less strict sense here, as “a description of an event that happened during project execution”. An example is “Because of frequent changes of the requirements, it was hard to see what consequences they would have. This affected the testing of the software.” Each such description, we will refer to as an “experience item”.

A way to categorize experience or knowledge is to look at where it is applicable, and how easy it is to transfer. Novins and Armstrong [5] have suggested the following framework for categorizing knowledge: Experience that is collected can be used in a setting that we can divide into two categories: *local* and *global*. If experience is usable only locally, it is not applicable for many people or projects. If it is globally usable, many people can benefit from it. We can also use two categories of how transferable knowledge is. If it is easily transferable, we say that it is *programmable*. If it is difficult to transfer, we say that it is *unique*. Then we get the Tab. 1.

Yet another way to classify experience would be according to the topic they are about, for example to which part of the development process they are related, to which organizational role, or to what tools or special work methods. We developed one set of categories that are relevant to the projects we will describe later:

Table 1. Categorization of experience according to applicability and transferability.

	Local	Global
Programmable	<i>Easy to transfer, but suits only in some situations.</i>	<i>Easy to transfer, and usable in many situations.</i>
Unique	<i>Hard to transfer, and only relevant in some situations.</i>	<i>Hard to transfer, but relevant in many situations.</i>

- Processes: Contract negotiation, estimation, planning, specification, design, implementation, testing, administration and maintenance.
- Actors: Customer, Project Manager, Management, Developer
- Technology: (no subcategories).

We will come back to how we applied these categorization frameworks in the discussion in section 4.

1.2 Collecting Experience from Projects

Now, how are we supposed to collect experience from completed projects? We first give an overview of a frequently used method, then introduce Postmortem Reviews, and explain how the effort in conducting such can be reduced to make “lightweight Postmortem Reviews”.

1.2.1 Experience Reports

A way to collect experience from a completed project is to write an “Experience Report”. This document is usually written by the project manager after the project is finished. The report follows a fixed template, which makes it possible to compare reports from different projects. In one of the companies we worked with, the report is divided into two parts: The first part gives an overview of all the facts and numbers from the project: Start and finish date, size of contract, labour used, deviation from estimated work size, the number of source lines-of-code developed, documents produced, and the activities that contributed most to the excess consumption. The second part of the report describes problems during project execution with proposal for improvement. For each phase of the project there is a Problem Description and Proposal for Improvements. This information is represented as text. In the other company they only have part two with problem definitions and proposed improvements.

These reports are usually not longer than 10-15 pages in one company we worked with, and about 4 pages in the other. About 50% is devoted to each part.

1.2.2 Postmortem Reviews

There are several ways to perform Postmortem Reviews. Apple has used a method [6] which includes designing a project survey, collecting objective project information, conducting a debriefing meeting, a “project history day” and finally publishing the results. At Microsoft they also put quite much effort into writing “Postmortem reports”, which are a bit more similar to what we have called “Experience Reports”. These contain discussion on “what worked well in the last project, what did not work well, and what the group should do to improve in the next project” [7]. The size of the resulting documents are quite large, “groups generally take three to six months to put a postmortem document together. The documents have ranged from under 10 to more than 100 pages, and have tended to grow in length”.

In a book about team software development, Watts Humphrey suggests a way to do postmortems to “learn what went right and wrong, and to see how to do the job better the next time” [8].

A problem with these approaches is that they are made for very large companies, who can spend a lot of resources on analysing completed projects. We work with medium-sized companies where 5-10 people usually participate in a project, ranging in size from about 8 to 50 manmonths. To suit this type of projects, we have developed a “lightweight” version of Postmortem Reviews.

1.2.3 Lightweight Postmortem Reviews

We have used Postmortem Reviews as a group process, where most of the work is done in one meeting lasting only half a day. We try to get as many as possible of the people who have been working in the project to participate, together with two researchers, one in charge of the Postmortem process, the other acting as a secretary. The goal of this meeting is to collect information from the participants, make them discuss the way the project was carried out, and also to analyse causes for why things worked out well or did not work out. A further description of what we did can be found in the “results” section.

Our “requirements” for this process is that it should not take much time for the project team to participate, and it should document the most important experience from the project, together with an analysis of this experience.

A description of another lightweight approach which seeks to elicit experience using interviews, and not a group process, is described by Schneider [9].

1.3 The Research Setting

Here we describe in what setting the research was carried out. We first introduce the research project we worked in, and then give a brief description of each of the companies and projects where we collected the empirical data for this paper.

1.3.1 The Research Project

The Process Improvement for IT industry (PROFIT) project is a Norwegian research project where researchers are working in tight cooperation with nine companies that develop software, with different process improvement initiatives. There are three main work packages: Process improvement under uncertainty and change, learning from experience, and process improvement through innovative technology and organization. The work which is reported here focuses on learning from experience. Some background information on earlier work on knowledge management systems within Norwegian software development companies has been published earlier [10].

1.3.2 Northern Software

Northern Software makes software and hardware for receiving stations for data from meteorological and Earth observation satellites. Since the company was founded in 1984, they have delivered turnkey ground station systems, consultancy, feasibility studies, system engineering, training, and support. Northern Software has been working with large development projects, both as a prime contractor and as a subcontractor. They possess a stable and highly skilled staff, many with masters degrees in Computer Science, Mathematics of Physics, and have what we can describe as a “engineering culture”. Approximately 60 people are working in the company, and the majority is working with software development. Projects are managed in accordance with quality routines fulfilling the European Space Agency PSS-05 standards, and are usually fixed price projects.

1.3.3 Southern Software

Southern Software is a software house specialising in advanced web-solutions, but not eBusiness. Examples are games, newspapers, customer services and resources, database access etc. Projects are usually small to medium sized. Larger projects are broken down in smaller packages by incrementally adding services to a web portal.

They have a heterogeneous staff; people with software as well as design background. Project teams often consist of many people with non-overlapping competence of webdesign- and programming, user interfaces, databases and transactions, software architecture, requirements and management. This means that communication costs are fairly high compared to overall project size.

The company recently started using Rational Unified Process (RUP) for some project parts (user requirements, management and testing). They recently assigned one full-time “knowledge manager” who receives all “knowledge harvest”-documents, user surveys and other project documents. This work has just started.

1.4 Scope

We could have looked at a lot of issues when comparing the two approaches discussed here. For example, we think that the Postmortem Reviews seemed to be

quite inspiring for the people who participated in the meetings, an effect you probably will not get with an Experience Report. However, we have limited the scope here to only discuss the results of the methods, that is, the written reports, and not the process of producing them, or any other effects they might have. What we will look for, is what kind of information we get out of each method.

2 Research Methods

The research performed is done in close collaboration with industry. We have participated in collecting experience from real software projects in a real environment, which means that we have little control of the surroundings. This is often referred to as action research [11, 12]. The benefit with this type of research is that the actual problems are very relevant to industry. A difficulty is that we have limited control over the surroundings, so the results might not be as generally applicable as when using for example experiments.

The material collected for this research is from two companies that we co-operated with in the PROFIT project. They were not selected arbitrarily; they were interested in working with the same topics that we were interested in. The projects we used as cases for lightweight Postmortem Reviews were selected by the companies. However, we asked them to select “typical” projects from their company, and also projects of a certain size.

The researchers have had two roles in this work: First, as a kind of process leader who have organized a meeting: Set the agenda in co-operation with industry and organized discussions. On the other hand, the researchers have been observers trying to document the process and the results of the meeting. In one company by using a tape recorder and transcribing important sections of the meeting, in another company by writing detailed minutes during the meeting.

When we analysed the material gathered, we had two reports, one experience report, and one post mortem review report from each company. An example part of an experience report is:

In this project the team members did not sit together, which complicated the communication between the members. In the last week of the project, however, the system track was placed together, which resulted in good communication and a stronger feeling of belonging to a team.

In our analysis, we would select sentences like the one underlined, and call them experience items. This was used in our later analysis as:

negative experience: physically separate

Another example is from a postmortem meeting, where one thing that was mentioned was:

Incremental development: Partial deliveries are motivating. Externally visible.

This was later documented in the report as:

The idea of incremental development, i.e. partial deliveries to the customer, worked very well. The customer became more aware of the project status, and was able to guide and enforce changes at an early stage.

Which was again summed up as an experience item:

positive experience: partial deliveries

Later, we would then do categorizations based on these experience items.

3 Results

Here we first present how we conducted lightweight Postmortem Reviews. Then we describe the contents of each of the reports from the Experience Report and the lightweight Postmortem Review, and give some examples of the experience that was collected. We only give examples from one company to save space, but will use results from both companies when we go on to discuss the differences between the results of the methods in section 4.

3.1 Lightweight Postmortem Review

We have used two techniques to carry out lightweight Postmortem Reviews. For a focused brainstorm on what happened in the project, we used a technique named after a Japanese ethnologist, Jiro Kawakita [13] – called “the KJ Method”. For each of these sessions, we give the participants a set of post-it notes, and ask them to write one “issue” on each. We usually hand out between three and five notes per person, depending on the number of participants. After some minutes, we ask one of them to attach one note to a whiteboard and say why this issue was important. Then the next person would present a note and so on until all the notes are on the whiteboard. The notes are then grouped, and each group is given a new name.

We use a technique for Root Cause Analysis, called Ishikawa or fishbone-diagrams to analyse the causes of important issues. We draw an arrow on a whiteboard indicating the issue being discussed, and attach other arrows to this one like in a fishbone with issues the participants think cause the first issue. Sometimes, we also think about what was the underlying reasons for some of the main causes and attached those as well.

Now, for the Postmortem Analysis meeting, we organized it with the following sections:

1. Introduction: First, we (the researchers) introduce the agenda of the day and the purpose of the Postmortem Review.

2. KJ session 1: We handed out post-it notes and asked people to write down what went well in the project, heard presentations, grouped the issues on the whiteboard, and gave them priorities.
3. KJ session 2: We handed out post-it notes and asked people to write down problems that appeared in the project, heard presentations, grouped the issues on the whiteboard, and gave them priorities.
4. Root Cause Analysis: We drew fish-bone diagrams for the main issues, the things that went well and the things that were problematic.

We used a tape recorder during the presentations, and transcribed everything that was said. The researchers wrote a Postmortem report about the project, which contained an introduction, a short description of the project that we analysed, how the analysis was carried out, and the results of the analysis. The result was a prioritised list of problems and successes in the project. We used statements from the meeting to present what was said about the issues with highest priority, together with a fishbone diagram to show their causes. In an appendix, we included everything that was written down on post-it notes during the KJ session, and a transcription of the presentation of the issues that were used on the post-it notes. In total, this report was about 15 pages long.

The day after the meeting, we presented the report to the people involved in the project to gather feedback and do minor corrections.

A comparison of this method to another way of performing lightweight Postmortem Review is discussed in another paper [14], where you also find information on the resources required.

3.2 Results from Lightweight Postmortem Review

One result from the KJ session was two post-it notes grouped together and named “changing requirements”. They are shown in the upper left corner of (some of) the results from the KJ process in Fig. 2.

Developer statements pertaining to this part of the KJ diagram:

“Another thing was changes of requirements during the project: from my point of view – who implemented things, it was difficult to decide: when are the requirements changed so much that things have to be made from scratch? Some wrong decisions were taken that reduced the quality of the software”.

“Unclear customer requirements – which made us use a lot of time in discussions and meetings with the customer to get things right, which made us spend a lot of time because the customer did not do good enough work.”

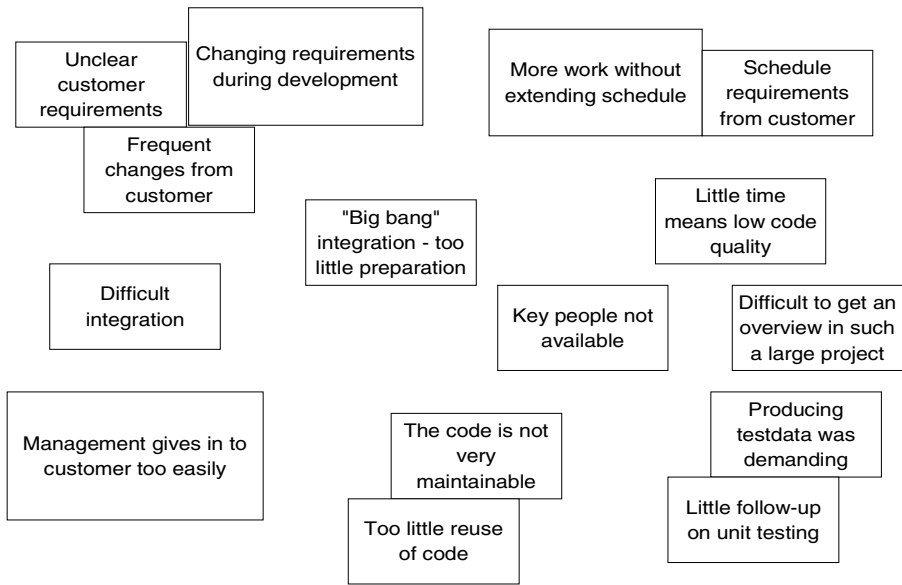


Fig. 2. Post-it notes showing some of the problems in a software development project.

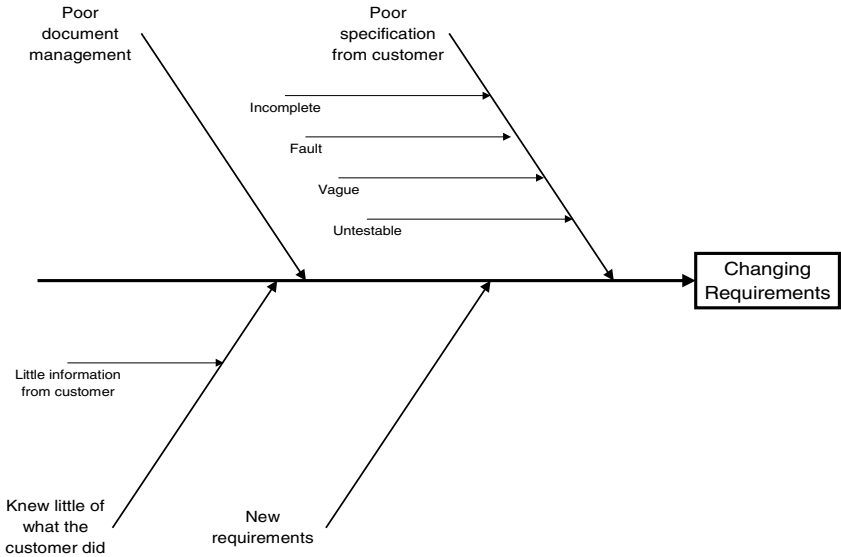


Fig. 3. Ishikawa diagram for “Changing Requirements”.

When we later brought this up again and tried to find some of the root causes for “changing requirements”, we ended up with the fishbone diagram in Fig. 3.

The root causes for the changing requirements, as the people participating in the analysis saw it, was that the requirements were poorly specified by the customer, there were “new requirements” during the project, and the company knew little of what the customer was doing. Another reason for this problem was that documents related to requirements were managed poorly within the company. In Fig. 3, we have also listed some subcauses.

In total, we found 21 experience items using this method at Northern Software, where nine were “positive”, and 12 were “negative”. At Southern Software, we found 18 “positive” and 8 “negative” experience items, making it a total of 26.

3.3 Experience Report

The project manager wrote the experience report. Of nine pages of information, five was introduction and description of the project, and four pages contained descriptions of “problems during project execution” with proposals for improvement.

An example of a problem is the “Architectural Design Phase”, which was described in the following way: “This phase should have been carried out in ten weeks according to the plan. We carried out this according to the plan, and was just two weeks late (this delay was introduced earlier in the project). We still got changes from (customer department) in this phase. The negotiations about the contract did not start before this phase was finished, and we worked for a while without an architecture proposal or a contract in a period. Much of the management work was done through the architecture design phase: Contract negotiation, revising schedules, etc”.

To solve the problems that appeared during this phase, the project manager has the following suggestions for improvement: “In total, this phase worked out ok. We were 300 hours behind schedule after this phase, which mainly came as a result of clarifying requirements from the customer. In the progress reports, I wrote that we should not give in to requests for changes in the estimates in the contract change notifications, or in the schedule. Experience indicated that we would get problems in the two next phases of the project, because the requirements were not stable, and the project was run according to (a standard process used in the company). I think we gave in to the customer in the negotiations about the schedule a bit easily, as well as the requirement on an intermediate delivery, which in a way worked as a template for the later negotiations. On the other hand, we got acceptance for the time estimates that we identified in contract change notification number 1 through 4.”

At Northern Software, we found in total 28 experience items, where 27 were about problems, and just one about an activity that went well. The company produced this report with no inference from the researchers, who started working with them at a later stage. At Southern Software, we found 26 experience items, where 11 were “positive” and 15 “negative”.

4 Discussion

Now, when we have seen a part of the reports from each of the two experience harvesting methods, can we say that there is any difference? Did the methods produce the same results? Could one method replace the other? To investigate this question, we studied the results in detail, and tried to group each of the experience items that were captured into one of the categories that we outlined in section 1.1.

We tried to apply the framework suggested by Novins and Armstrong, and to see if the experience could be said to be either *unique* or *programmable*, and *local* or *global*. Here, almost everything seemed to fit in the category *programmable* – the experience seemed to be pretty easy to transfer. To us, it looked like the experience would be valid for the whole company, that is, *global*. So with this framework, we could not distinguish between the results from the two methods.

We further found that almost all of the experience from both reports in both companies seemed to be of a kind that we can describe as “Problem understanding” [15, 16] – they were not as detailed as to use as a guideline, and we found little hard “facts” apart from in the introduction section in the Experience Report. Most of the information seemed to help “develop a better comprehension of a particular problem”, and would be usable for “problem understanding”.

Then, we tried to look at the topics that the experience was about. We made a list of *processes* in use in the companies, *actors* that would have different responsibilities, and also a category for *technology*, for experience related to tools, platforms and products. Using this type of categorization, and allowing one experience item to be related to both an actor, a processes or “technology”, we found that the experience divided into the categories given in Fig. 4.

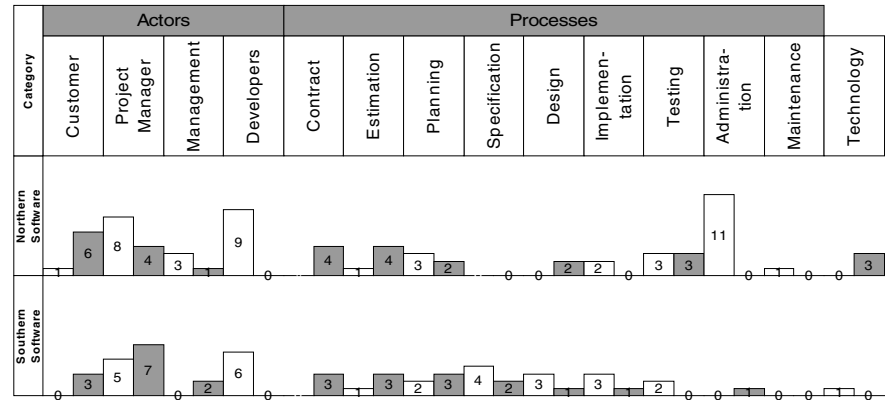


Fig. 4. Experience according to categories. Results from the Lightweight Postmortem Review in white and from the Experience Report in grey.

We see that most of the categories have experience from each of the methods, although the number varies. If we try to see which categories that have experience only from Postmortem Reviews, from both of the methods, and experience only

from the Experience Report, we find that the majority of experience categories are covered by both methods (6 categories). Postmortem Reviews covers one more (4) category than the Experience Reports (3) for Northern Software, but it is the other way around in the results from Southern Software. So it seems that the methods have covered slightly different issues.

To investigate this further, we examined the experience we had put in each category in more detail. Would these experience turn out to be the same, or different issues related to the same things in the same category? The results of this examination are given in Tab. 2. We have used the same set of categories as in Fig. 4, but replaced the name of each category with a number to save space. The number of categories that are “similar” means that “that number of the same experience items were found with both methods”. The “total” figure refers to the sum of experience found with both methods.

From Tab. 2, we see that relatively few experience items are about the same issue. Only 5 experience items were found with both methods of a total of 69 experience items in Northern Software (some were put in several categories, so the sum of the “totals” in the table will be larger). At Southern Software, we found only three experience items with both methods, of a total of 49 experience items.

Table 2. Similarities of experience from each of the categories. “Total” is the total number of experience items in each category. Results from postmortem reviews are abbreviated PM, and experience report ER.

Category		Actors				Processes									
		1	2	3	4	5	6	7	8	9	10	11	12	1	3
Northern	PM	1	8	3	9	0	1	3	0	0	2	11	1	0	
	ER	6	4	1	0	4	4	2	0	2	0	0	0	3	
	Total	7	12	4	9	4	5	5	0	2	6	11	1	3	
	Similar	2	1	2	0	0	0	0	0	0	0	0	0	0	
Southern	PM	0	5	0	6	0	1	2	4	3	2	0	0	1	
	ER	3	7	2	0	3	3	3	2	1	0	1	0	0	
	Total	3	12	2	6	3	4	5	6	4	2	1	0	1	
	Similar	0	0	0	0	0	1	0	2	0	0	0	0	0	

So why is this? Why is it that we did not find a very large overlap in the results of the methods? One reason might simply be that by using a group process to elicit experience, we get to view what happened through several “different eyes”. A reason in the Southern Software case might be that the content of the lightweight Postmortem Review was known when the Experience Report was written. But it was the other way around in the Northern Software case.

Another thing we found, independently of the categorization frameworks, is that most of the experience from the Experience Report at Northern Software were about problems, whilst we intended to get 50% about problems and 50% about successes in the Postmortem Reviews. A reason for this might be that the Experience Report is used more to explain what went wrong than the Postmortem Reviews, which had a

more precise goal of capturing experience that might be useful for others. At Southern Software, we did not find this in the Experience Report.

A difference we noted in Southern Software, was that the lightweight Postmortem Review would assign an experience to a situation and role, whereas the Experience Report would have a more managerial view of the experience, relating to the overall process. For example, enforcing formal change requests from the customer resulted in better cost control and planning from the management point of view. However, the developers considered that the main improvement was that they were relieved of direct, and frequent, customer interruptions.

5 Conclusions

Now we will sum up the conclusions we can draw from the discussion:

- The two methods find very little overlapping experience: We found more experience related to implementation, administration, developers and maintenance with the lightweight Postmortem Review. Whereas with the experience reports, we found more experience related to contract issues, design and technology.
- The experience items found with both methods seem to be usable for most projects within the companies, and seem to be quite easy to transfer.
- The two methods seem to find experience that can be used for problem understanding.

In all, it seems that both methods elicit information that are similar in style, but which is related to a bit different topics. If you are interested in issues related to the core processes of software development, lightweight Postmortem Reviews seems to be a better method than Experience Reports. If you are more interested in relations to the customer, Experience Reports seem to be a better choice.

Acknowledgements

We would like to acknowledge colleagues in the PROFIT project for providing a stimulating research environment, as well as our contact persons in Northern and Southern software. We are very grateful to Stefan Biffl at the University of Vienna, Reidar Conradi at the Norwegian University of Science and Technology as well as the anonymous reviews, for helpful comments on this paper. Furthermore, we are grateful to Geir Kjetil Hanssen at Sintef Informatics who managed part of the Lightweight Postmortem Review at Southern Software, and who made important contributions to the analysis. This work was supported by the Norwegian Research Council under project 137901/221.

References

- [1] K. M. Wiig, *Knowledge Management Methods*: Schema Press, 1995.
- [2] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Experience Factory," in *Encyclopedia of Software Engineering*, vol. 1, J. J. Marciniak, Ed.: John Wiley, 1994, pp. 469-476.
- [3] M. Polanyi, *Personal Knowledge - Towards a Post-Critical Philosophy*. London: Routledge and Kegan Paul, 1958.
- [4] PERFECT Consortium, "PIA Experience Factory, The PEF Model,," ESPRIT Project 9090 D-BL-PEF-2-PERFECT9090, 1996.
- [5] P. Novins and R. Armstrong, "Choosing your Spots for Knowledge Management," *Perspectives on Business Innovation*, vol. 1, pp. 45-54, 1998.
- [6] B. Collier, T. DeMarco, and P. Fearey, "A Defined Process For Project Postmortem Review," *IEEE Software*, vol. 13, pp. 65-72, 1996.
- [7] M. A. Cusomano and R. W. Selby, *Microsoft Secrets - How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*: The Free Press, 1995.
- [8] W. S. Humphrey, "The Postmortem," in *Introduction to the Team Software Process, SEI Series in Software Engineering*. Reading, Massachusetts: Addison Wesley Longman, 1999, pp. 185-196.
- [9] K. Schneider, "LIDs: A Light-Weight Approach to Experience Elicitation and Reuse," presented at Second International Conference on Product Focused Software Process Improvement, PROFES 2000, Oulu, Finland, 2000.
- [10] R. Conradi and T. Dingsøyr, "Software experience bases: a consolidated evaluation and status report," presented at Second International Conference on Product Focused Software Process Improvement, PROFES 2000, Oulu, Finland, 2000.
- [11] D. J. Greenwood and M. Levin, *Introduction to Action Research*: Sage Publications, 1998.
- [12] D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM*, vol. 42, pp. 94-97, 1999.
- [13] R. Scupin, "The KJ Method: A Technique for Analyzing Data Derived from Japanese ethnology," *Human Organization*, vol. 56, pp. 233-237, 1997.
- [14] T. Stålhane, T. Dingsøyr, N. B. Moe, and G. K. Hanssen, "Post Mortem - An Assessment of Two Approaches," submitted to European Conference on Software Process Improvement, EuroSPI, Limerick, Ireland, 2001.
- [15] C. W. Choo, *The Knowing Organization : How Organizations Use Information to Construct Meaning, Create Knowledge, and Make Decisions*: Oxford University Press, 1998.
- [16] R. S. Taylor, "Information Use Environments," presented at Progress in Communication Science, 1991.

A Tool for Managing Software Development Knowledge

Scott Henninger and Jason Schlabach

Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
{scotth, jschlaba}@cse.unl.edu

Abstract. Software development is a knowledge intensive activity involving the integration of diverse knowledge sources that undergo constant change. Most approaches to knowledge management assume that information exists and is readily sought by software developers, usually through a search tool. In this paper, an approach is presented that actively delivers information through a rule-based system that matches system requirements to experience-based knowledge embedded in work breakdown structures. A reuse-based methodology based on an organizational learning process is used to capture and organize knowledge as it is created. The combination of tool and methodology work together to capture characteristics of individual projects and tailor processes to meet diverse and emerging software development needs. The tool and methodology are demonstrated using two examples of how this technique can be applied

1 Knowledge Management for Software Development

Software development is a knowledge-intensive activity involving the integration of diverse knowledge sources that undergo constant change. The knowledge required to successfully develop software systems ranges from programming techniques to the “tool mastery burden” [6] to the critical role of application domain knowledge [10] and the diverse nature of software development processes and techniques [27]. In addition, the knowledge is under constant pressure from fluctuating requirements and technology advances, causing knowledge volatility and further pressures on the knowledge mastery burden.

The management of this knowledge and how it can be brought to bear on software development efforts has received little attention in the software engineering research community. Most software engineering methodologies view the project as the object of study, assuming that the project begins with a blank sheet of paper and leaves a documentation trail only to assist the maintenance phases of the project. This leaves little basis that the organization, or the community as a whole, can use to understand and learn from previous development efforts. Tools, techniques, and methods are needed that capture knowledge in a continuous knowledge acquisition process and disseminated in a form that can be brought to bear on subsequent development efforts.

Traditional methods to capture and organize knowledge use repository technology coupled with a search engine. While this technique addresses a real need for information searching, it tacitly assumes that people know what information exists and can articulate their information need properly. Although this assumption can be questioned in general information retrieval settings [5], it is particularly dubious in software engineering, where studies have shown that functions and methods are regularly re-implemented, often differing only in name [11]. Tools and techniques are needed that not only allow search for information, but also actively alert software developers to the existence of potentially relevant knowledge.

In the following sections, we present a tool and methodology for software development knowledge management that draws on past development experiences and actively presents information of potential relevance to development efforts. First, the overall system infrastructure and methodology are presented, followed by two examples of how this technique can be applied. We finish with a discussion of what has been learned from this research and future directions.

2 Building an Organizational Repository of Experiences

An organizational learning approach to software development [22] captures project-related information during the creation of individual software products which is then disseminated to subsequent projects to provide experience-based knowledge of development issues encountered at a software development organization. These principles have been demonstrated through an exploratory prototype, named BORE (Building an Organizational Repository of Experiences) [17].

Early BORE research focused on developing tools to support the creation of case-based repositories [17]. Evaluations of these prototypes [16] revealed issues often found when adopting software tools [23]. In particular, we found that BORE activities were regarded as a pure documentation effort with few perceived benefits, and was therefore seen as ancillary to the immediate goal of producing a working software product. Because BORE was not intimately tied to these goals, people regarded using the system as supplemental and not part of the critical path. Despite our efforts to present the organizational learning approach [17, 19], the simple repository tools provided by BORE were insufficient to allow people to draw on past experiences. Tools were needed that actively presented relevant information, guides people through the methodology, and collects and disseminates this knowledge as part of the development process.

The work described here draws on these experiences and extends the BORE work by coupling organizational memory tools with software process tools based on a work breakdown structure. The studies also demonstrated the need to guide the use of the tool with a corresponding methodology that addresses how existing information is brought to bear on new projects, and how new knowledge is captured to meet the emerging needs of software development efforts.

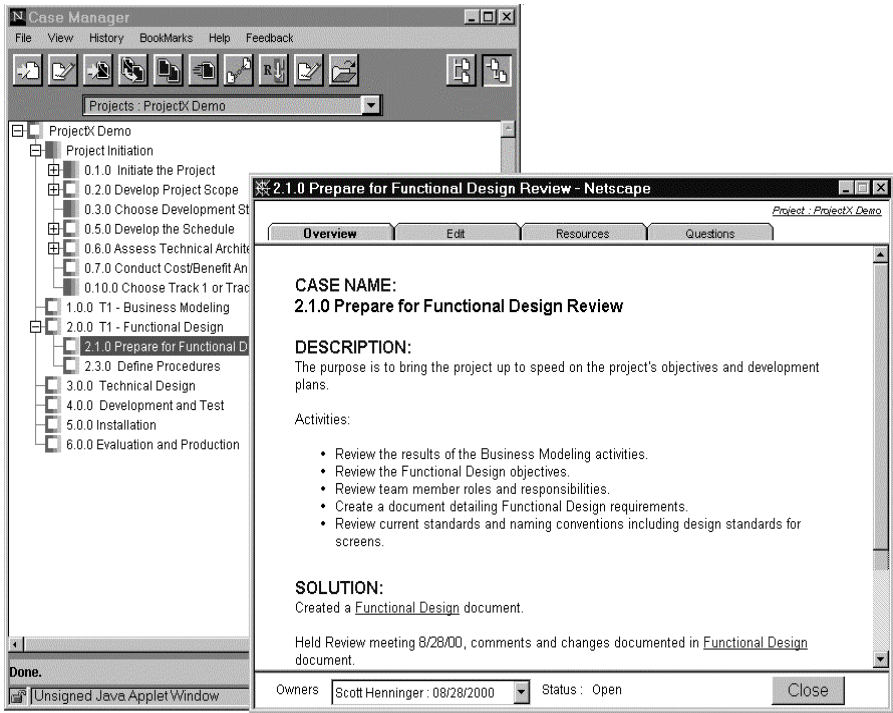


Fig. 1. BORE Case Manager and a case

2.1 The BORE Prototype

The BORE prototype is a Web-enabled application using a three tiered architecture consisting of HTML and Java AWT for rendering the interface, Java for application logic, and relational database back-end. It can be accessed through a Web <http://cse-ferg41.unl.edu/bore.html> (log in as 'guest')¹

The main interfaces for BORE are shown in Fig. 1. The Case Manager, shown to the left in Fig. 1, displays a hierarchical arrangement of project activities. In the figure, a project named "projectX demo" has been chosen from the list of resources that can be displayed in the Case manager's tree view display. Each case contains project-specific information as shown in the window to the right in the figure, which was obtained by double-clicking on the activity named "Prepare for Functional Design Review" in the project hierarchy.

Cases are used in a case-based decision support approach [25], and describe situation-specific solutions to software development activities. Cases consist of a description to a problem, a solution statement, and some ancillary information, such

¹ Results are best if using Netscape Communicator 3.1 or greater or Internet Explorer 5.0 or greater. Java and Javascript must be enabled.

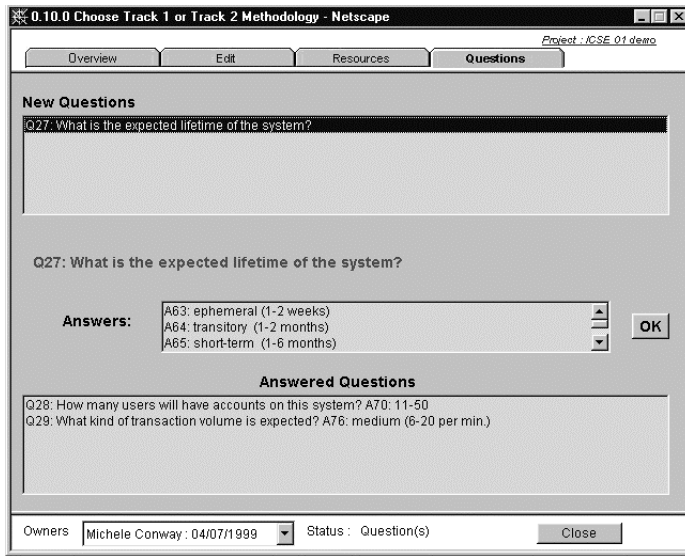


Fig. 2. The Options tab.

as related cases, etc. For example, in [Fig. 1](#), the case describes a specific activity of preparing for a functional design. The description was copied when the case was added to the project. The solution field is then used to document project-specific information accumulated while executing the activity. In addition, a color-coded icon is used in the Case Manager to depict the status of a case (active, resolved, etc.), thus providing a way to visually summarize a project's status.

2.2 BORE Knowledge Domains

Bore divides its repository into domains. Domains are independent knowledge realms that consist of a set of domain *cases* defining activities and domain *rules* that define the context under which a process is applicable to a development effort. Currently, projects belong to a single domain, which is chosen when a project is created in BORE. All subsequent project activities will use the cases and rules defined for that domain. This supports scalability and allows organizations to partition development activities into domains that address the needs of diverse units and/or product lines.

Domain Cases define the space of possible activities for projects within a given domain, structured in a work breakdown hierarchy. They define standard activities that have proved useful for situations encountered by projects within the domain. The cases describe some of the problems or necessary steps that must be taken to ensure that the process is properly followed, and can be updated by a project to reflect specific activities the project follows. For example, an activity may dictate that the corporate standard login screen is used for an application. A project trying to follow this step may find that using the password database required a couple of work-arounds

to fit their specific architecture. The project’s case would reflect these problems and document how the problems were solved in the “Solution” field of BORE cases (see Fig. 1).

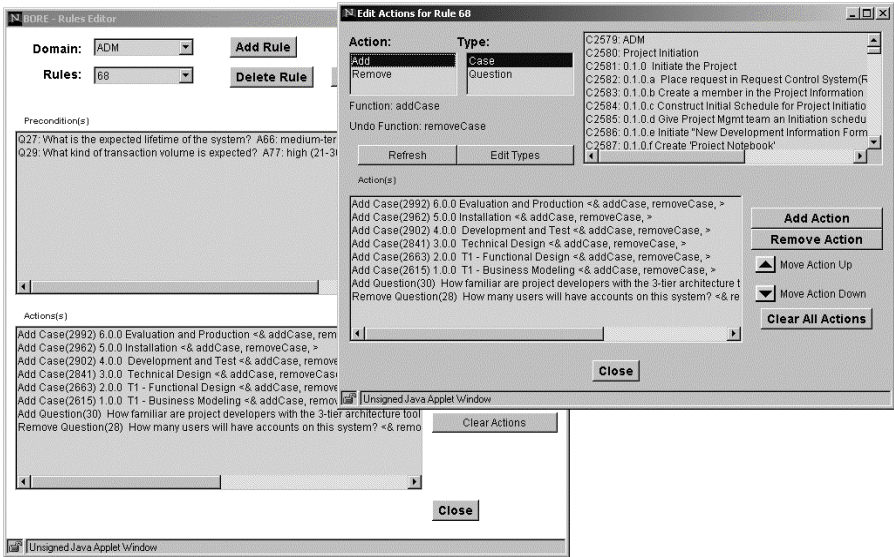


Fig. 3. The Rules Manager.

Domain Rules are used to tailor the overall development process to the specific needs of different projects. Internally, BORE uses a simple forward chaining production system implemented using an SQL database to represent rules. Rules consist of a set of preconditions and actions. Preconditions are represented by question/answer pairs. Questions are associated with the Options tab of a case (see Fig. 2). Cases that have options assigned to them appear in the Case manager with a ‘?’ in the status icon. Options are chosen by project personnel to tailor the organization’s standard process to individual project needs. For example, Fig. 2 shows a session in which two questions have already been answered (i.e. the options were chosen) and a third has been chosen, revealing the possible answers. These options address high-level project requirements that may impact which activities are chosen for the project. Options can be designed to address requirements at any level, from business concerns to development tools to reusable components.

When all preconditions of a rule evaluate to true, the rule “fires,” causing a set of defined actions to be executed. The core actions in BORE can remove questions from the question stack (the New Questions pane of Fig. 2), add a question to the question stack, or add a case to a project. In addition, new action types can be created by attaching a Java method to the action. BORE supports rule backtracking that allows rule actions to be undone when a precondition changes in such a way that rule should no longer be fired. Therefore, all action types supporting backtracking need to develop both an action and an undo method.

2.3 Creating and Modifying Domains

During project execution, knowledge in a BORE domain is refined in two ways. First, a new case is instantiated for each process element assigned to the project. This case will document how the project met the requirements of the given process element. This will also provide a resource for subsequent development efforts because one can look at the cases for all projects assigned a given activity. The second modification to the repository involves creating new domain rules and cases to reflect project activities that should become part of the overall domain. For example, if a project determines that a Java Plug-in is necessary when developing applets that must run both cross-platform and cross-browser, a review team may decide that the project's action are setting a precedent of significant value to other projects. Modifying the domain to reflect this new knowledge involves 1) creating a new rule, and 2) creating new domain cases, and 3) associating new options with domain case(s).

Creating Rules. Rules are edited by selecting a rule and editing the associated preconditions and actions. Fig. 3 shows the interface for editing rules. Clicking on the Edit Preconditions button pops up another window (not shown) that lists all questions in the domain. For each precondition clause a question is chosen along with the answer (as stated earlier, all questions in a rule have an implicit AND relationship). In our example, the rule will fire when the question in the preconditions box is answered 'yes'. This will cause two actions to fire, one that adds a new question and another that adds a case to the current project. New questions and associated answers can be added by clicking the Edit Questions button on either the Window in Fig. 3 or the Edit Preconditions window (not shown). Cases containing questions are designated in the BORE Case Manager with a green question mark icon.

Actions are added to the rule through the window to the right in Fig. 3 that lists the possible action types. Once a type is chosen, the defined actions are listed and can be selected for inclusion in the rule. New actions and action types can be added by creating a link to an existing Java object with methods for executing and undoing the action.

Creating Domain Cases. Domain cases are edited in Case Manager and are treated just like any other project hierarchy in BORE. The domain librarian (someone with permissions to edit domains) selects the domain needing editing and adds, moves, deletes, and edits domain cases as needed. Domain cases are added using the buttons at the top of the Case Manager and edited with the Edit tab on the case.

Attaching Rules to Domain Cases. The questions used to choose options can be attached to domain cases by editing the Options tab in the domain case. Any questions chosen will be displayed in the Options tab when the case is created for a project.

Implementing BORE in an organization will require having personnel trained in rule-based systems. Rule systems can become nearly as complex as programs themselves, and people will need to ensure that the rules perform the appropriate actions and that rules are not inconsistent, contradictory, or incomplete. The benefits arising from these additional costs, as described later, is a formalized definition of an organization's process that can be tailored to individual project needs, thus reducing

the burden on project personnel to become software process experts before using the organizations defined standard.

3 Examples Using BORE

Domain rules and cases are used to define software processes at any level of detail, from high-level development processes to specific corporate standards, such as which login screen should be used. The level of desired detail is left entirely to the designer of the domain. To demonstrate this versatility, we present two examples of how BORE can be used for software development activities. The first shows how BORE can be used to define a standard development methodology that can be tailored to specific project needs. The second shows how a set of parameterized templates can be used to define Oracle configuration files and SQL scripts.

3.1 A Defined, Extensible SDM

One application of BORE is to create a tailorable Standard Development Methodology (SDM) that can serve as a focal point for an organizational learning process. This approach is similar in focus to software process technology, but is less concerned with automating parts of the process [8, 15], and more concerned with work breakdowns and providing resources when they are needed. In spite of advances in software process technology, most organizations continue to document the majority of their standard development practices in monolithic SDM documents. Hypertext versions of the documents on Web-based Intranets are becoming popular, but solve none of the problems inherent with finding information and applying it to a specific software development effort.

An inherent problem with SDMs is the tendency to try to accommodate diverse development needs with an SDM that is stated at a very high level and lacks the detail that can truly guide a project. Industry standards, such as CMM and ISO do little to address this issue, and may in fact contribute to watering down processes to create universally applicable development methodologies. There also needs to be specific guidelines, examples, and detailed procedures that help people create the design. I.e., to be successful, the SDM must provide valuable resources for development efforts.

To reduce these problems, BORE can create an on-line SDM using rules to create multiple paths through the SDM. The paths can be designed to accommodate the different kinds of projects typically encountered in the organization [30]. The question/answer preconditions of BORE rules serve as a specification of the conditions under which a specific activity should be applied to a project. Options can be added to cases that allow projects to choose amongst alternative processes. The example in Fig. 2 is taken from the software development standard of a large transportation company that defined two major tracks for their process, one for developing applications that will be deployed across the enterprise, and another for more localized and prototyping efforts.

One way to design this kind of SDM would be to have each project answer a series of questions at project initiation. We find this approach unappealing for a number of

reasons. First, we want BORE to support a number of different levels of decision making, from choosing whether the project is enterprise-wide or local to what kind of database should be used all the way to choosing reusable code, such as logon screens and associated logic or back-up and recovery schemes. Secondly, if a system spans these levels of support, and indeed it must to be called an organizational memory, it would be impossible to anticipate the answers to all questions at project initiation.

Therefore, we used a more iterative strategy. When a project begins, an initial set of cases for a project can be defined any of which can have options associated with it. Choosing some options may further break down an activity into constituent activities. Since any case can have options, these can implement further detailed breakdowns. For example, an initial pass may break a system into architecture-level elements, such as a user interface. Embedded in the user interface activities can be options to decide which interface widgets should be used, what choice methods are applied and etc. This multi-step process allows projects to incrementally expand each part of the project's process as they gain the knowledge necessary to address process issues [30]. In this way, the domain can be designed to tailor the overall SDM to its needs when it is able to address the issues involved.

3.2 An Extensible Wizard Using Parameterized Templates

Another knowledge intensive activity in the development process is the creation of databases to support application programs. In many development organizations, domain experts, often named Database Administrators, or DBAs for short, are called upon for this often laborious task. In this application of BORE, we designed an extensible wizard to embody DBA knowledge to create network configuration files and SQL scripts for creating tables in an Oracle database. The SQL script is a file of SQL statements. The network configuration files allows for complexities of the network, such as the Net8 network layer used by Oracle, to be hidden from applications and the database. We used BORE to generate an entry in the configuration file that resides on the client (in a client/server architecture) to allow a simple way for the client-side application to communicate with the database server.

Wizards are a convenient mechanism for directing people to relevant information when the needs are well-known. Microsoft Word provides wizards for common tasks such as creating envelopes, Web pages, legal pleadings, etc. These Wizards direct users through a set of choices, and the desired object (such as a MS Word document or a database table) is created automatically by the system.

While traditional wizards have been applied to technical domains, such as component generators for data structures [4], the static nature of Wizards needs to be addressed before the method can be used as a software development tool. For example, while Oracle provides wizards that aid in the creation of database objects, such as tables and constraints on those tables, there is no way to extend this functionality to create other database objects or embody domain-specific knowledge such as local conventions for naming and organizing tables. It would be useful if the wizard could capture knowledge specific to an organization and disseminate that knowledge. For example, a banking or loan organization may frequently develop systems that need a lender table. The wizard can then allow the user to choose the default lender table, rather than go through all of the generic questions about which

```
?service_name?.world = (DESCRIPTION =
  (ADDRESS_LIST = (ADDRESS = (PROTOCOL
    = TCP) (Host = 129.93.33.120) (Port =
    1521))) (CONNECT_DATA = (SID = ORCL)))
```

Fig. 4. A template for creating Oracle network configuration file entry.

fields, data types and sizes, and constraints are necessary for the lender table. This approach facilitates enhanced reuse of knowledge and code.

Developing an Extensible Wizard with BORE. To create a wizard for creating Oracle database tables, we designed a BORE domain that constructs working database components for a simple class of address book applications. This application domain applies knowledge from the second author's experiences with a student lending organization.

This was accomplished by creating string templates that contain variables, creating BORE rules that performs text replacement in the templates, placing the resultant strings in an appropriate file (such as network configuration file or SQL script), and using the resulting files to create database objects on an Oracle server. An example template is shown in Fig. 4. It is entered through the Rule Manager as a simple text string. The template defines variables by surrounding a variable name by question marks, such as the "service_name" variable in Fig. 4. Through the facility for adding new action types, a Java object was created with methods to search the text for variables and replace the string with the values of variables defined by BORE rules.

In addition to string replacement in the templates, methods were developed to create actions that communicate with the database in order to create new schemas and new tables. This application requires that the new database tables are created in an existing Oracle database. Defaults, such as standard tablespaces (logical storage areas), for this database are already established and included in BORE rules. An Oracle database server runs on the same machine as the BORE server and is accessed through system calls.

Creating the Database Tables. The BORE wizard interface is used the same way as with the SDM. That is, the user makes choices within the Options tab of the BORE case window. In addition to the simple single answer selections shown in Fig. 2, Fig. 2 can select multiple answers or type a text answer. The latter is used in conjunction with a rule that associates a rule variable with some text, such as names and other items that cannot be predefined in the domain. In the student lending domain, the service name is example text that needs to be filled in by users.

All variables do not necessarily *have* to be manually typed by the user. For example, at the specific institution we developed this system for, all development databases resided on one machine while all production databases resided on another machine. We used a question that determined whether the new tables were to be created on the development machine or the production machine. BORE rules then determined the IP address and port number.

If the user should go back and change the answer to a question, the appropriate rules are backtracked and new rules fire. With the example of our SQL script file, backtracking a rule that created a SQL statement will comment out the statement with a timestamp. If the rule that created the file is backtracked, then the file is renamed with a timestamp. This process of renaming files and commenting out statements allows for a history of the actions to be kept.

A Feasibility Study. To assess the feasibility of this approach, we asked three test subjects to develop the database component described above for the address book application. Two subjects were experienced Oracle developers from a student loan organization with no BORE experience. The third was a BORE researcher/developer with no Oracle experience and no experience in the student lending problem domain. We asked the subjects to both build the component using whatever traditional development method they would ordinarily use, and to build the component using the BORE system.

The task involved creating a database schema with three tables (LENDER, SCHOOL, PERSON) where each table is stored in one of the standard (already existing) tablespaces, and a way for other system components to communicate with the database component.

All of the test subjects were able to build the database component using the BORE wizards in a matter of minutes. Only one of the test subjects was able to successfully create the database component using their traditional method, and it is significant to note that this database expert needed to be prompted by the experimenter to insert a database directive to complete the component. The BORE researcher/developer, as expected, was not able to create the database component without the use of BORE.

None of the test subjects were able to expand the rule base for the new tablespace requirement mentioned in the above paragraph. The issue here seems to be less on the nature of using a rule-based system, and more on the need for improvements in BORE's rule editing capabilities, a fact that has been acknowledged in the past and needs further work.

4 Related Work

The general goal of this approach is to create knowledge management tools that are more proactive in delivering information to software developers than typical repositories and search engines. One way this can be accomplished is to create a tailorable process that provides context-sensitive information to software development efforts. Another way is to create extensible wizards that anticipate desired features, allowing for extensions when unanticipated features are needed.

The BORE tool and methodology is flexible enough to split the gap between overly-restrictive development methodologies and ad-hoc software development practices to fit the needs of software development organizations as they evolve. Currently, the industry standard is to define a SDM and then ignore it in its entirety or follow it enough to justify it to certification authorities. We wish to turn these procedures and software processes in general into resources that truly supports the development process as it is actually practiced, while adding necessary degrees of

formal procedures to ensure high-quality products. This involves not only defining a process, but also using feedback from projects to refine and improve its procedures.

The concept of experience factories [1-3] has many common themes to this work, although little in the way has been done to create interfaces and CASE tools to support the concept. The QIP approach in TAME [3] framework is designed to develop and package experiences to facilitate reuse within the organization. Basili et al. advocate the use of metrics and quantifiable goals to create an improvement strategy and address controlling the content, structure and validity of the knowledge [1]. While these metrics will become necessary for an organizational learning approach to succeed, our focus and contribution thus far has been to provide a support environment for this kind of approach in a framework that allows for decisions support for choosing appropriate processes and evolution of processes to continuously improve the organization's best practices.

Thus far, most research on software processes has focused on defining the process. Approaches include high-level strategies such as the waterfall, spiral, and prototyping models, methods for combining process elements [30], and universal process models such as the CMM [31] or ISO 9000. A significant contribution of this work is to define not only the process, but how the process evolves with the changing needs of the development organization, a phenomena that process technology is only beginning to explore [8]. BORE domains are seen as a "seed" that evolves as it is used [14]. In addition, the process can be defined at many levels of detail, allowing projects to adopt the process at an appropriate level of detail. In essence, this is more of a form of knowledge editing [32] than process programming [9, 30], which is more concerned with integrating tools and documents to automate development activities, although elements of both issues are present.

This approach also has some roots in the design rationale field [7, 13, 26, 28, 29]. Similar to the organizational learning approach, the motive for capturing design rationale is to avoid repeating common pitfalls or re-discussion of design decisions by describing the alternatives debated and decisions made for a given effort. Many schemes, from the simple of Procedural Hierarchy of Issue structures [7, 13] to more complex structures designed to capture logical relationships for computation [26] have been devised. All have the same basic structure of a set of hierarchically arranged questions posed to flesh out issues. Alternatives and rationale for the alternatives can be attached to the questions to document discussion paths. BORE extends these techniques by incorporating knowledge other than design rationale and creating rule-based techniques for identifying when issues are applicable.

There are also common themes with the Designer Assistant project at AT&T, which created an organizational memory system to ensure conformance to the usage of a specific piece of complex functionality in a large switching system [33]. In this setting, the design process was modified to include a trace of the Designer Assistant session as part of a design document. The appropriateness of the designer's choices and adequacy of the advice given by Designer Assistant are discussed during software design reviews. If the advice is found to be lacking, designers begin a formal change process to update the knowledge. Utilizing a combination of existing and new organizational processes to place use of Designer Assistant into development practices ensures that the knowledge will evolve with the organization. The

observation that “technology and organizational processes are mutual, complementary resources” [33] has served as a guiding principle for this work.

5 Conclusions and Future Work

Knowledge management for software development is more than repositories and search engines. It also requires actively delivering information during the development process and ongoing process of capturing project experiences. Using such “active” knowledge management techniques can prevent the duplication of efforts, avoid repeating common mistakes, and help streamline the development process. Similar projects have shown that such a system will be used by development personnel, provided it contains relevant, useful and up-to-date information [33]. This mandates a strong tie between technology and process in which using the technology must become part of routine work activities. Such an approach will succeed to the extent that developers are rewarded in the short term for their efforts.

The contribution of this research is to develop a combination of tools and techniques that turn current development practices into *living* documents designed to evolve and improve through use, drawing on the collective knowledge of the organization.. As the repository accumulates through principled evolution of the knowledge domain, it improves to able to handle a wider range of circumstances [24], while evolving toward answers to problems that fit the organization’s technical and business context. The real question is not whether the repository is “correct” in some objective sense, but rather whether less mistakes are repeated and better solutions adopted when using the repository.

These principles were demonstrated by applying our approach to SDM processes and wizards for developing database components for address book applications. As demonstrated in our small empirical study, the knowledge embedded in these tools can be invaluable not only for novice users, but experts as well.

Future Directions. Aside from refining the normal quirks of a research prototype, a number of improvements are needed to take the next step in evaluations of BORE. It was clear from our brief study that the rule interface of BORE needs some improvements. In particular, there needs to be ways to find a rule given some behavior (such as a case being added to a project) or other attributes. Currently, users need to page through the rules one-by-one to find a rule with the desired precondition or action.

More tools are needed that support creating domain rules and ensuring that the rules are consistent and no contradictory. Visualizations and aggregate rules will prove useful, and we are looking into using agents and critics to flag knowledge base inconsistencies and gaps [32] and ensure that complete and consistent processes are assigned to projects.

The extensible wizard work was designed as a step towards applying BORE to component-based software engineering environments. We are currently working on using BORE to select and configure components in the JavaBeans platform to create software applications using the extensible wizards described here.

Early BORE evaluations indicate progress has been made on our overall goals of creating a medium for flexible software process definitions and turning SDM documents into a resource that supports the development process as it is actually practiced [20, 21].

The BORE system is beginning to move out of the prototyping phase and become a production-quality system, which will enable evaluations in realistic software development settings. Within the coming months, we hope to deploy it in a few development organizations, including NASA Goddard [18], some local small organizations, and the Software Design Studio at the University of Nebraska-Lincoln [12]. Through these evaluations, we hope to gain an increased understanding in the overall approach and how knowledge-based tools such as BORE can be used to improve software development practices.

Acknowledgements. We gratefully acknowledge the efforts a number of graduate students that have helped develop BORE, particularly Kurt Baumgarten, Michelle Conway, and Roger Van Anel. This research was funded by the National Science Foundation (CCR-9502461 and CCR-9988540).

References

1. V. R. Basili, G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, vol. 1, pp. 53-80, 1992.
2. V. R. Basili and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, pp. 303-316, 1991.
3. V. R. Basili and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, vol. 14, pp. 758-773, 1988.
4. D. Batory, G. Chen, E. Robertson, and T. Wang, "Design Wizards and Visual Programming Environments for GenVoca Generators," *Transactions on Software Engineering*, vol. 26, pp. 441-452, 2000.
5. N. Belkin, "Helping People Find What They Don't Know," *Comm. of the ACM*, vol. 43, pp. 58-61, 2000.
6. F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 20, pp. 10-19, 1987.
7. E. J. Conklin and K. Yakemovic, "A Process-Oriented Approach to Design Rationale," *Human-Computer Interaction*, vol. 6, pp. 357-391, 1991.
8. G. Cugola, "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models," *IEEE Transactions on Software Engineering*, vol. 24, pp. 982-1000, 1998.
9. B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," *Communications of the ACM*, vol. 35, pp. 75-90, 1992.
10. B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, pp. 1268-1287, 1988.
11. P. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard, "LaSSIE: A Knowledge-Based Software Information System," *Communications of the ACM*, vol. 34, pp. 34-49, 1991.
12. S. Dunbar, S. Goddard, S. Henninger, and S. Elbaum, "Bootstrapping the Software Design Studio," *Fifth Annual National Collegiate Inventors and Innovators Alliance National Conference*, Washington, DC, pp. 180-188, 2001.

13. G. Fischer, J. Grudin, A. Lemke, R. McCall, J. Ostwald, B. Reeves, and F. Shipman, "Supporting Indirect Collaborative Design With Integrated Knowledge-Based Design Environments," *Human-Computer Interaction*, vol. 7, pp. 281-314, 1992.
14. G. Fischer, R. McCall, J. Ostwald, B. Reeves, and F. Shipman, "Seeding, Evolutionary Growth and Reseeding: Supporting the Incremental Development of Design Environments," *Proc. Human Factors in Computing Systems (CHI '94)*, Boston, MA, pp. 292-298, 1994.
15. V. Gruhn and J. Urbainczyk, "Software Process Modeling and Enactment: An Experience Report Related to Problem Tracking in an Industrial Project," *Proc. 20th International Conference on Software Engineering*, pp. 13-21, 1998.
16. S. Henninger, "Capturing and Formalizing Best Practices in a Software Development Organization," *The Ninth International Conference on Software Engineering and Knowledge Engineering (SEKE '97)*, Madrid, Spain, 1997.
17. S. Henninger, "Case-Based Knowledge Management Tools for Software Development," *Journal of Automated Software Engineering*, vol. 4, pp. 319-340, 1997.
18. S. Henninger, "Software Process as a Means to Support Learning Software Organizations," *Twenty-fifth Annual NASA Software Engineering Workshop*, Greenbelt, MD, 2000.
19. S. Henninger, "Supporting Software Development with Organizational Memory Tools," *International Journal of Applied Software Technology*, vol. 2, pp. 61-84, 1996.
20. S. Henninger, "Tools Supporting the Creation and Evolution of Software Development Knowledge," *Proceedings of the Automated Software Engineering Conference*, Lake Tahoe, NV, pp. 46-53, 1997.
21. S. Henninger, "Using Software Process to Support Learning Software Organizations," *1st International Workshop on Learning Software Organizations (LSO 1999)*, Kaiserslautern, FRG, 1999.
22. S. Henninger, K. Lappala, and A. Raghavendran, "An Organizational Learning Approach to Domain Analysis," *17th International Conference on Software Engineering*, Seattle, WA, pp. 95-104, 1995.
23. C. C. Huff, "Elements of a Realistic CASE Tool Adoption Budget," *Communications of the ACM*, vol. 35, pp. 45-54, 1992.
24. J. L. Kolodner, *Case-Based Reasoning*: Morgan-Kaufman, San Mateo, CA, 1993.
25. J. L. Kolodner, "Improving Human Decision Making through Case-Based Decision Aiding," *AI Magazine*, vol. 12, pp. 52-68, 1991.
26. J. Lee, "Design Rationale Capture and Use," *AI Magazine*, vol. 14, pp. 24-26, 1993.
27. M. Lindvall and I. Rus, "Process Diversity in Software Development," *IEEE Software*, vol. 17, pp. 14-18, 2000.
28. A. Maclean, V. Bellotti, R. Young, and T. Moran, "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human-Computer Interaction*, vol. 6, pp. 201-251, 1991.
29. T. Moran and J. Carroll, "Design Rationale: Concepts, Techniques, and Use," Hillsdale, NJ: Lawrence Erlbaum Associates, 1996.
30. L. Osterweil, "Software Processes are Software Too," *Ninth International Conference on Software Engineering*, Monterey, CA, pp. 2-13, 1987.
31. M. C. Paulk, B. Curtis, M. Chrissis, and C. V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, vol. 10, pp. 18-27, 1993.
32. L. Terveen and D. Wroblewski, "A Collaborative Interface for Browsing and Editing Large Knowledge Bases," *National Conference of the American Association for AI*, Boston, MA, pp. 491-496, 1990.
33. L. G. Terveen, P. G. Selfridge, and M. D. Long, "Living Design Memory' - Framework, Implementation, Lessons Learned," *Human-Computer Interaction*, vol. 10, pp. 1-37, 1995.

Starting Improvement of Requirements Engineering Processes: An Experience Report

Marjo Kauppinen and Sari Kujala

Helsinki University of Technology, Software Business and Engineering Institute,
P.O. Box 9600, FIN-02015 HUT, Finland

{marjo.kauppinen, sari.kujala}@hut.fi

Abstract. Defining and managing customer requirements is becoming increasingly important in product development. Many software organizations are interested in improving their requirements engineering processes, but they do not know how and where to begin. This report describes the experiences of two Finnish organizations that have started to develop their requirements practices systematically. To guide these development activities, software process improvement procedures and guidelines published in the literature were adjusted and applied. The most important lesson learned was that introducing requirements engineering can require a change of culture and not merely a change of process and technology. The change of culture firstly requires that product development personnel fully understand the reasons for documenting requirements from a customer's point of view. Secondly, they must commit to defining and managing customer requirements systematically.

1 Introduction

An increasing number of organizations are interested in improving their requirements engineering (RE) processes. The first question these organizations must answer is how to start developing their existing RE practices. In theory, requirements engineering applies proven principles, techniques, languages and tools [3]. In practice, organizations need guidance on how and where to start the improvement of their RE processes. The literature offers process improvement approaches, success factors of software process improvement cases and requirements engineering guidelines to be used in RE process development.

If organizations want to gain lasting and visible benefits from process improvement, they need to adopt a systematic approach. The IDEAL model [11] and the ISO/IEC 15504 standard [8] are two well-known approaches to software process improvement. Both can be considered as representing an advanced approach to process improvement because applying them successfully requires effort and expertise from organizations. Many organizations do not have sufficient resources and experience to use the advanced approaches as such. This report describes a simple process improvement procedure that combines tasks from both the IDEAL model and the ISO/IEC 15504 standard.

Organizations that desire to improve their RE processes can also use the lessons learned from successful software process improvement. This report summarizes six guidelines that are based on the factors identified as critical to software process improvement.

While the simple process improvement procedure and the six guidelines provide guidance on how to improve software process, they do not define where to begin the development of the RE processes. Sommerville et al. have defined top ten RE practices that all organizations should implement [18]. They further recommend that organizations start their RE process improvement program by implementing these top ten guidelines [18].

The main contribution of this report is the lessons learned from the first improvement experiences of the RE processes in two Finnish organizations that wanted to develop their RE processes systematically. The authors supported organizations to deploy the simple process improvement procedure, the six software process guidelines, and the top ten RE practices. This experience report is based on data collected through observations, informal conversations, formal interviews, official meetings, and document studies.

The experience report is structured as follows. In the next section, we describe the simple process improvement procedure used in this study. The third section summarizes the six process improvement guidelines that have been published in different articles. The top ten RE practices recommended by Sommerville et al. are briefly presented in Section 3. Our two cases are described in Section 4 and the lessons learned from these two cases are explained in Section 5. Finally, we summarize the results of the two case studies.

2 Process Improvement Procedure

Organizations can use various models and standards to guide the improvement of their RE processes. The IDEAL model and the ISO/IEC 15504 standard are approaches for software process improvement. Both approaches include a procedure that helps an organization improve its existing software practices systematically.

The IDEAL model, which has been developed by the Software Engineering Institute (SEI), consists of a five-phase improvement procedure [11]. Each phase can include up to ten activities. The ISO/IEC 15504 standard is also known as SPICE. The process improvement procedure of the ISO/IEC 15504-7 standard involves eight phases, each of which can contain up to five activities [8].

Both the IDEAL model and the 15504 standard can be considered as representing an advanced approach to process improvement because applying them successfully requires expertise and resources from organizations. For example, the IDEAL model recommends that at least one full-time person is assigned to software process improvement [11]. Many organizations do not have sufficient resources and process improvement experience to use the advanced approaches as such. We have defined a simple process improvement procedure for small and medium-sized organizations. This procedure involves five phases and combines activities from both the IDEAL model and the ISO/IEC 15504 standard. The phases are presented in Figure 1. In this

report, we concentrate on the first two phases and the activities of them are summarized in Table 1.

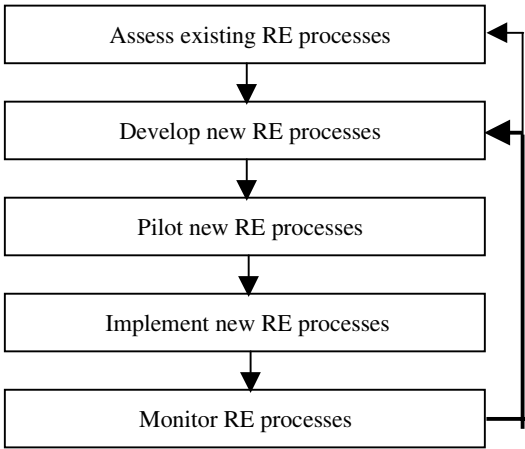


Fig. 1. Phases of the process improvement procedure

In the assessment phase, the organization’s current state is analyzed to define a baseline for the RE process improvement. The purpose of the assessment is to provide information based on which organizations can set realistic improvement targets and plan practical improvement actions. The output of the phase is an assessment report and a preliminary project plan.

In the development phase, process improvement teams create solutions to the problems and the challenges discovered during the assessment. The solutions can be for example a new practice, method, or template. The purpose of the development is to document the new RE process. The documentation ensures that the developed solutions can be rolled out throughout the organization.

Table 1. Activities of the first two phases of the process improvement procedure

Phase	Activities
Assessing the existing RE processes.	<ul style="list-style-type: none">• Planning the assessment.• Collecting data.• Analyzing data.• Reporting the assessment results.• Planning improvement actions.
Developing new RE processes.	<ul style="list-style-type: none">• Forming project improvement teams.• Defining new RE processes.• Documenting the new RE processes.• Reviewing the new RE process documents

After the improvement teams have developed a new RE process, it is piloted in real projects, which ensures that the developed RE processes are practical and product development personnel find them useful. After the piloting phase, the new RE process is systematically deployed throughout the organization. The purpose of the deployment is that product development personnel are aware of the new RE process and are able to define and maintain requirements according to the new practices.

The last phase of the process improvement procedure involves monitoring the impact of the improvement efforts. Measurement is the only way to provide quantitative data about the results of the improvement actions. By implementing measurement, organizations can follow up how the RE practices are developing and what the return on investment is. Based on the analysis of the measurement data, the organization can initiate a new process improvement project.

3 Guidelines for RE Process Improvement

Software engineering researchers have found critical factors for successful software process improvement. The following six guidelines summarize a set of the most frequently identified success factors. Organizations can use these general guidelines as a checklist when they start to improve their RE process.

1. *Set goals for process improvement* [4], [5], [8], [11], [14], [16], [19]: The objectives of a process improvement program should be clearly stated and understood, and expressed using measurable process goal [8].
2. *Link process improvement goals with business objectives* [4], [5], [8], [11], [14], [15], [20]: It is important to emphasize productivity, quality and cycle time and to avoid process for its own sake [4]. The process improvement goals should be aligned with the business goals of the organization. Without this alignment, long-term improvement will usually not be sustained [15].
3. *Ensure management commitment and support* [2], [4], [5], [6], [10], [13], [14], [15], [19], [20]: Management commitment is required from all levels [4]. Executive leadership is so critical that an improvement project should not be initiated without it [2]. One of the primary reasons why improvement efforts fail is that middle management resists the introduction of improvements [2]. In practice, management commitment and support means that management funds and provides staff for process improvement [10].
4. *Manage process improvement as a project* [2], [8], [13], [14], [16], [20]: The fundamental lesson underlying the IDEAL model is that process improvement needs to be managed like any other project [2]. Gaining visibility and tracking progress one must view process development as a project. This means applying structured planning and tracking mechanism [14]. It is also important to take resource assignment seriously [13] and not to underestimate efforts required for process improvement actions [16].
5. *Use teams formed from practitioners to improve the process* [2], [4], [5], [8], [10], [11], [18], [19], [20]: Software process improvement is a team effort [8]. A characteristic of successful improvement projects is that the improvements are designed by those who must adopt them [2]. Practitioners, not outside process experts, should define process [4]. The process improvement effort was staffed by people who indicated interest in the effort, then worked with

their supervisors to allocate the time and resources to make their participation successful [10].

- 6. *Convert assessment findings into realistic improvement actions* [8], [18], [20]: Moving from assessment to improvement planning and implementation is often difficult for many organizations [20]. Organizations must strike a balance between introducing too many new practices at once and introducing too few new process changes which have limited effect [18]. It is desirable to include some improvement actions which yield clear short-term benefits, particularly if the organization is new to process improvement, in order to encourage acceptance of the process improvement program [8].

The process improvement procedure and the general improvement guidelines offer guidance on how to start developing processes. Organizations interested in improving their RE practices can use the general procedure and guidelines, but they additionally need more specific guidance for RE process improvement. Table 2 summarizes the top ten RE guidelines recommended by Sommerville et al. [18]. These ten RE practices are so important that they should be implemented in all organizations and organizations should start their process improvement program by implementing them [18].

Table 2. Top ten RE guidelines [18]. Low-cost practice should involve less than 10 days of effort, moderate-cost involves 10-100 days of effort, and high-cost involves more than 100 effort-days [18].

RE practice	Development effort
Define a standard document template	Moderate-high
Make the document easy to change	Low
Uniquely identify each requirement	Very low
Define policies for requirements management	Moderate
Define standard templates for requirements description	Moderate
Use language simply, consistently and concisely	Fairly low
Organize formal inspections	Moderate
Define validation checklist	Low-moderate
Use checklists for requirements analysis	Low-moderate
Plan for conflicts and conflict resolution	Low

4 Cases

Both case organizations are product development units of about 150 persons. Organization A develops products that have both real-time embedded and interactive components. The systems of Organization B are interactive. Both development units joined a RE research project at the beginning of 1999. Here, we report our experience of the first two phases of the process improvement procedure. The improvement projects are still in progress and are the subject of a longitudinal study.

4.1 Assessment Phase

The improvement projects started up with a meeting in which the needs for RE improvement and the process improvement procedure were discussed. Both organizations had already invested in their RE practices, but they had not organized the development work as a project. Both organizations decided to start an appropriate project and to use the five-phase improvement procedure.

In the start-up meeting, the goals for the assessment phase were defined. The main goal of the assessment was to set a baseline for long-term monitoring, because the organizations wanted to measure the progress of the coming improvement actions. Setting the baseline was particularly important, because neither organization had measured its existing RE practices. The organizations additionally wanted to use the assessment results for benchmarking, which was made possible by use of the REAIMS maturity model developed by Sommerville et al. [18].

In the second meeting, the case organizations planned the assessment phase. Firstly, they defined the scope of the assessment and then selected experienced persons from the various product development units to participate in the interviews that were based on the REAIMS maturity model. The purpose of the REAIMS interviews was to define quickly the current state of the RE process. The organizations also decided that the RE practices of three on-going projects would be analyzed. The analysis was done by interviewing the person who had been responsible for defining the requirements and by assessing the requirements documents of the projects. The purpose of these in-depth interviews and document analysis was to understand deeply the current state of the existing RE practices.

Based on the data collected through the interviews and the document studies, the maturity level of the RE processes of the organizations was defined. We found projects that had produced a good requirements document, but the RE processes of the organizations were at the lowest level of the REAIMS maturity model. The assessment results can be summarized in the following way:

- Most of the requirements documents were based on narrative text and it was difficult to recognize individual requirements from the text. In some projects, all requirements had already been assigned a unique identifier, but they were put into a long list and relations between requirements had not been documented.
- Most requirements described design solutions and internal behavior of the system and only some of them were defined from a customer's point of view.
- The status of the requirements was not followed, and requirements documents were not updated although requirements changes were common.
- Systems were not tested based on requirements documents.

The product development engineers of Organization A described the existing situation in the following way: "Customer requirements are not defined systematically. The requirements definition is not a routine procedure and it requires a lot of mental effort. Practices how to utilize a customer requirements document during a product development project are missing." The personnel of Organization B mentioned the following problems related to their RE practices: "How to collect requirements directly from customers and what customers are really doing with the

systems developed by us are essential questions. Customer requirements are not described clearly. It is difficult to find requirements from the text. Requirements should be presented as exactly as possible." Both the formal assessment results and the comments of the practitioners show that the culture of the systematic customer requirements definition and management was missing in the case organizations.

The assessment results were documented and the report was delivered to all the participants of the assessment and the manager of the improvement project. The contents of the report were also presented in a workshop. Organization A had a half-day and Organization B a whole-day session. The members of the RE process improvement project participated in the workshop in Organization A. In Organization B, the managers of the different product development departments attended the session. The purpose of the workshops was to analyze the assessment results, set goals for RE process improvement, and select improvement areas. The analysis of the assessment results was lightweight. The assessors presented the main findings and the participants commented and made questions, and this took about two hours in Organization A and three hours in Organization B.

Both organizations experienced difficulties in setting goals for RE process improvement. In Organization A, the participants of the workshop discussed goals, but objectives could not be stated clearly and were not documented in the project plan. In Organization B, one of the managers had had the experience that the goal setting for process improvement is time-consuming and frustrating, and he proposed that objectives would not be set during the workshop. The other participants accepted this proposal. The process improvement group later defined the following goals: 1) to define the requirements subprocess as a part of the existing product development process and 2) to improve the quality of requirements engineering. The group also decided that the quality of the new requirements process would be measured with the metric "number of requirements changes".

4.2 Development Phase

The development phase started by forming improvement teams for the selected improvement areas. Organization A decided to designate groups to the following six areas: 1) System requirements, 2) platform architecture design, 3) splitting system requirements to components, 4) requirements elicitation and analysis process, 5) requirements validation process and 6) requirements definition and management processes. The number of team members varied from two to four, and eighteen persons altogether participated in the development phase. The team members were product development engineers and project managers, who were doing process improvement work at the same time as they were members of product development projects. They were expected to reserve an hour's effort a week for the RE improvement project.

Organization B formed three improvement teams, one for requirements elicitation procedure, one for requirements analysis and one for requirements change management and traceability. The number of team members varied from three to four. The total number of persons who participated the development phase was fifteen. The team members were product development engineers and project managers, who were

involved in process improvement work at the same time as they were members of product development projects.

Each of the improvement teams organized their work in a way that best suited its members. All teams reported their results and possible problems in the follow-up meetings. The main goals of these follow-up sessions were to ensure that the teams were working in the same direction and to give feedback from the improvement work.

The improvement teams of Organization A decided to produce the following deliverables: a system requirements template, a traceability matrix, a template for component interfaces, guidelines for requirements elicitation, guidelines for requirements validation and finally requirements definition and management process description. Organization B decided to provide a requirements process definition that includes an elicitation procedure, analysis procedure and change management procedure definitions. Table 3 summarizes the improvement actions of both organizations.

Table 3. The improvement actions selected by the case organizations.

RE area	Improvement actions of Organization A	Improvement actions of Organization B
Requirements document	<ul style="list-style-type: none"> Define a standard template for requirements 	<ul style="list-style-type: none"> Define a standard template for requirements
Requirements elicitation	<ul style="list-style-type: none"> Record requirements source Use scenarios to elicit requirements 	<ul style="list-style-type: none"> Record requirements source Record requirements rationale
Requirements analysis	-	<ul style="list-style-type: none"> Prioritize requirements
Requirements representation	<ul style="list-style-type: none"> Define standard templates for requirements description 	-
Requirements validation	<ul style="list-style-type: none"> Define guidelines for formal requirements inspection Define checklists for requirements validation 	-
System modeling	<ul style="list-style-type: none"> Document the links between requirements and system models 	<ul style="list-style-type: none"> Document the links between requirements and system models
Requirements management	<ul style="list-style-type: none"> Uniquely identify each requirement Define policies for requirements management Define change management policies 	<ul style="list-style-type: none"> Uniquely identify each requirement Define policies for requirements management Define change management policies Define traceability policies

The improvement teams of Organization B used its existing standard template for documenting developed procedures. The procedure documentation of each improvement team was also reviewed formally. The teams updated their procedure definitions based on the review and the project manager combined these definitions into a single document. Because Organization A did not have a standard template to document their product development procedures and guidelines, the improvement project produced a set of different kinds of documents. The deliverables of the improvement teams were reviewed informally.

5 Lessons Learned

We identified five factors that influenced the success of the RE process improvement at the beginning of the improvement projects. The lessons learned from the two cases are:

- Introducing requirements engineering can require a change of culture.
- It is essential to convert assessment results into realistic improvement actions.
- Setting measurable goals for RE process improvement is an important yet demanding task.
- Management support is essential.
- It is important that improvement teams have experienced practitioners from different departments.

These lessons are based on observations, informal conversations, formal interviews, official meetings and document studies and they are described in more detailed in the following sections.

5.1 Change of Culture

Introducing requirements engineering can require a change of culture. The change of culture means two things: Firstly product development personnel need to understand the importance of customer requirements and secondly they must commit to defining and managing customer requirements systematically.

Our main finding related to the differing understanding of requirements. People did not mean the same thing when they discussed requirements. In most cases, requirements were defined from a technical point of view and they were in effect design solutions. Both of the case organizations had recognized the need for defining requirements from a customer's point of view, but the requirements documents for the most part described the internal behavior and technical details of the system.

The detailed assessment of the existing RE practices revealed that the organizations did not define and manage requirements systematically. The product development projects did not have a culture that was sufficiently precise and committed to requirements. The product development personnel had not thoroughly understood why it is important to document customer requirements clearly and to maintain customer requirements document throughout development projects. This means that introducing requirements engineering requires a change of culture and not merely a change of process and technology.

5.2 Realistic Improvement Actions

It is essential to convert assessment results into realistic improvement actions. The process improvement project must align the improvement efforts with available resources. In addition, it is important to include actions that yield short-term benefits, particularly if the organization is new to RE process improvement.

The main problem was the difficulty in estimating how much effort and calendar time was required for the improvement actions. Some process improvement teams

became frustrated and began to lose interest in improving the RE process because the improvement actions took longer than expected. One solution to this problem is to prioritize actions and begin with the basic RE practices. We recommend the following four improvement activities for organizations that are beginning to invest in their RE processes:

1. Define what the term requirement means in your organization.
2. Define standard templates for representing individual requirements.
3. Define the purpose of the requirements documents.
4. Define a standard structure for requirements documents.

Organizations that start to improve their existing RE practices need to answer two basic questions that are 1) what a requirement is and 2) how it can be presented. A short description of a requirement can help people to use the term requirement consistently and it reduces the likelihood of misunderstanding. Standard structures make requirements easier to write and read.

When the organization has decided what a requirement is and how it is presented, it needs to decide what the contents of a requirements document is and the purposes for which it can be used. Different types of groups can use requirements documents. First of all, a requirements document guides software engineers in designing and testing. It can additionally be used in project planning and follow-up. Moreover, a requirements document can provide material for user manual writers and marketing personnel.

Table 4 provides approximate effort estimations for the four actions we recommend for organizations that want to start developing their RE processes. The effort estimations are based on our experience and development costs provided by Sommerville et al. [18].

Table 4. Four basic improvement actions for organizations that want to start improving their RE processes. Very low-cost improvement action should involve a 2-3 days of effort, low-cost involves less than 10 days of effort and moderate-cost involves 10-100 days of effort.

Improvement action	Development effort
Define what the term requirement mean in your organization	Very low -low
Define standard templates for representing individual requirements	Low-moderate
Define the purpose of requirements documents	Very low
Define a standard structure for requirements document	Moderate

5.3 Measurable Goals for RE Process Improvement

Setting measurable goals for RE process improvement is an important but a demanding task. First, improvement goals can motivate the members of the process improvement teams. It is important that people understand reasons for investing in the RE process. Clear and shared goals can also guide the improvement teams to keep the target in mind and help them to work in the same direction. If process improvement goals are measurable, it is easier to evaluate the success of the improvement efforts.

The main reason for the difficulties in setting measurable goals was that the organizations did not have quantitative data from their existing RE practices.

However, it is unrealistic to expect organizations that are just starting to improve their RE process to have measured their RE practices.

Based on the experiences of the two cases, we recommend that organizations that are starting to improve their RE practices define a vision statement for RE process improvement. "The purpose of the RE process is to guarantee that customers and users are satisfied with our products" is an example of a vision statement. After organizations have implemented measurement and they have quantitative data, they can set measurable goals for RE process improvement.

5.4 Management Support

Management support is essential. Management involvement in RE process improvement is particularly important because improvement actions can concern not only product development, but also such organizational units as a company's sales, sales support, and marketing. Managers can also guarantee that a process improvement project has enough skilled personnel and enough time for the work. Process improvement can be frustrating because results can often be seen in the long term. Therefore, teams need feedback from management that they are doing important work.

The difficulty with management support was that the managers were busy. Medium and large organizations can have several process improvement projects in progress simultaneously and managers can have limited time for each improvement project.

Based on the experiences of the two cases, we recommend that managers help a project group to set improvement goals and ensure that the RE process improvement project has sufficient resources. During the project, management can provide visible support and encouragement in follow-up meetings.

5.5 Improvement Teams

It is important that improvement teams have experienced practitioners from different departments. Experienced practitioners bring knowledge of the existing RE practices. They can also ensure that improvement actions fulfil the needs of their departments and the actions are practical. Another benefit associated with having experienced practitioners from different parts of the organization is that these persons can spread information about the improvement actions in their own departments.

The main problem was that practitioners were busy with product development work and had limited time to spend on process improvement work. An effective way to save practitioners' time was to involve a process expert in the improvement team. The process expert can organize development sessions and document the results of the team.

Teams that had busy members also had difficulties in finding time to get together. This meant that improvement efforts took several months' calendar time although each team member spent only a couple of days on the process improvement. A simple solution for this problem was to organize development sessions regularly and fix the

dates sufficiently in advance. The practitioners found one three hour session every three weeks an effective way to organize the improvement work.

6 Conclusions

This paper describes the lessons learned in two Finnish organizations that have started to develop their RE practices systematically. To guide these development activities, the process improvement procedures and guidelines published in the literature have been adjusted and applied. The goal of this experience report is to offer guidance on how and where to start the improvement of RE processes and we want to emphasize what lessons other organizations can learn from the two cases.

The main lesson for organizations that want to start improving their requirements practices is that introducing requirements engineering can require a cultural change. Product development engineers frequently define requirements from a technical point of view. A major challenge is that documenting requirements from a customer's point of view requires developers to change their way of thinking. Another challenge is how to get product development projects to define and manage customer requirements systematically. Two other case studies report the cultural change towards systematic customer requirements management to be a success factor for RE process improvement [7], [9]. Furthermore, Basili et al. report that the most effective process changes are those that leverage the thinking of developers [1].

In order to increase the understanding of the importance of customer requirements, we recommend organizations to start the improvement of their RE processes by defining the meaning of the term "customer requirement" and the purpose of customer requirements documents. Based on our experience, such improvement actions help product development engineers to document requirements from a customer's rather than a technical point of view.

We further recommend organizations to define standard templates for representing individual requirements and a standard structure for customer requirements documents. Such improvement actions help product development personnel to define and manage customer requirements systematically. Kamsties et al. similarly suggest that a requirements document is the basic requisite for all other RE activities [12]. Furthermore, Sawyer et al. classify standard templates as a basic RE practice for organizations developing market-driven software products [17].

We found that the success factors of the software process improvement cases are additionally essential in RE process improvement. It is important to set measurable goals for RE process improvement, to ensure management support, to use improvement teams formed from experienced practitioners and to select realistic improvement actions. Such process improvement guidelines are effective, but organizations can face difficulties if they are just beginning to invest on RE.

In order to be able to set measurable goals for RE process improvement, organizations should have quantitative data from their existing RE practices. The problem is that organizations just starting RE process improvement do not often have existing measurement data based on which goals can be set. Another challenge is how to get busy managers and experienced practitioners to support RE process

improvement. The problem with selecting a set of realistic improvement actions is that it is difficult to estimate efforts and time required for RE process improvement actions.

The improvement projects of the case organizations are still in progress and are the subjects of a longitudinal study. In the near future, we will focus on the following two research questions: What are the characteristics of a good requirement and a good requirements document? One important challenge in the long term is to evaluate the impact of the improvement actions.

References

1. Basili, V., Zelkowitz, M., McGarry, F., Page, J., Waligora, S., Pajerski, R.: SEL's Software Process - Improvement Program. *IEEE Software*, Vol. 12, Issue 6 (1995) 83-87
2. Curtis, B.: Software Process Improvement: Methods and Lessons Learned. *Proceedings of the 19th International Conference on Software Engineering* (1997) 624-625,
3. Davis, A. M., Hsia, P.: Giving Voice To Requirements Engineering. *IEEE Software*, Vol. 11, Issue 2 (1994) 12-15
4. Diaz, M., Sligo, J.: How Software Process Improvement Helped Motorola. *IEEE Software*, Vol. 14, Issue 5 (1997) 75-81
5. Haley, T.: Software Process Improvement at Raytheon. *IEEE Software*, Vol. 13, Issue 6 (1996) 33-41
6. Humprey, W., Snyder, T., Willis, R.: Software Process Improvement at Hughes Aircraft. *IEEE Software*, Vol. 8, Issue 4 (1991) 11-23
7. Hutchings, A., Knox, S.: Creating Products Customers Demand. *Communications of the ACM*, 38 (5) (1995) 72-80
8. Information technology – Software process assessment – Part 7: Guide for use in process improvement, Technical report, ISO/IEC TR 15504-7:1998(E) (1998)
9. Jacobs, S.: Introducing Measurable Quality Requirements: A Case Study. *Proceedings of the 4th IEEE International Symposium on Requirements Engineering* (1999) 172-179
10. Johnson, A.: Software Process Improvement Experience in the DP/MIS Function: Experience Report. *Proceedings of the 16th International Conference on Software Engineering* (1994) 323-329
11. McFeeley, B.: IDEAL: A User's Guide for Software Process Improvement. Handbook CMU/SEI-96-HB-001. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PE, USA (1996)
12. Kamsties, E., Hörmann, K., Schlich, M.: Requirements Engineering in Small and Medium Enterprises. *Requirements Engineering*, Vol. 3, No. 2 (1998) 84-90
13. Kautz, K., Hansen, H., Thaysen, K.: Applying and Adjusting a Software Process Improvement Model in Practice: The Use of the IDEAL Model in a Small Software Enterprise. *Proceedings of the 22nd International Conference on Software Engineering* (2000) 626-633
14. O'Hara, F.: European Experiences with Software Process Improvement. *Proceedings of the 22nd International Conference on Software Engineering* (2000) 635-640
15. Paulk, M. et al.: The Capability Maturity Model: Guidelines for Improving the Software Process. Carnegie Mellon University, Software Engineering Institute, Addison Wesley Longman, Reading, Massachusetts (1997)
16. Sakamoto, K., Nokakoji, K., Takagi, Y.: Toward Computational Support for Software Process Improvement Activities. *Proceedings of the 20th International Conference on Software Engineering* (1998) 22-31,

17. Sawyer, P., Sommerville, I., Kotonya, G.: Improving Market-Driven RE Processes. Proceedings of International Conference on Product Focused Software Process Improvement (PROFES'99), Oulu Finland (1999) 222-236,
18. Sommerville, I., Sawyer, P.: Requirements Engineering – A Good Practice Guide. John Wiley & Sons, New York (1997)
19. Tanaka, T., Sakamoto, K., Kusumoto, S., Matsumoto, K., Kikuno, T.: Improvement of Software Process by Process Description and Benefit Estimation. Proceedings of the 17th International Conference on Software Engineering, pp. 123-132, 1995.
20. Zahran, S.: Software Process Improvement: Practical Guidelines for Business Success. Addison Wesley (1998)

A Family-Oriented Software Development Process for Engine Controllers

Karen Allenby, Simon Burton, Darren Buttle, John McDermid,
John Murdoch, and Alan Stephenson

Rolls-Royce University Technology Centre in Systems and Software
Engineering, Department of Computer Science, University of York, York, YO10
5DD, UK

Mike Bardill and Stuart Hutchesson
Rolls-Royce Plc, Control Systems, SinC-3, P.O. Box 31, Moor Lane, Derby,
DE24 8BJ, UK

Abstract. This paper presents a software engineering process that includes family-based aspects for aero-engine control software. The process uses a “family capability” derived from family analysis of the embedding system. This analysis allows reuse capability to be designed into the process by exploiting prior knowledge of the differences between members of an engine family. The process described follows the traditional software lifecycle. However, the nature of the stages is modified to incorporate the use of the family capability, essentially to allow systematic application of both compositional and generational reuse throughout the development process. The process described was evaluated on a trial project to develop, and subsequently modify, a thrust reverser system for a small aero-engine. Initial results show that this process can reduce effort for both initial and subsequent developments through the use of the family capability.

Introduction

Aero engines are developed as families at the mechanical level - that is, individual family members are derived from a common conceptual design by adaptation, e.g. scaling a turbine to increase the engine thrust. By contrast, the control software for family members does not benefit from an equivalent “family capability”. In this paper we present the results of a collaborative research programme undertaken by Rolls-Royce Plc and the Rolls-Royce University Technology Centre (UTC) in systems and software engineering aimed at addressing this situation.

The intention of the programme was to develop a software engineering process that could exploit the commonality between controllers to provide a “family capability” for control system software. This type of development process stems from a history of reuse-driven approaches, building on some of the first successes in describing families [Parnas76]. Later work achieved success by focusing on narrow domains [Neighbors84] and introduced modern domain analysis concepts [Prieto-Díaz90].

Research in this area has tended to produce automated techniques to locate components and integrate them into a software product. However, in the safety-critical domain, the cost of building certifiable software tools is a deterrent to the use of automated methods.

Prior to this programme, tactical reuse had been observed between projects, but there had been little systematic reuse. Previous work in the UTC addressed this observation [Hill-et-al94]. However, this level of reuse was not sustained, and subsequent projects saw significant change traffic. In part, the difficulties in sustaining reuse arose because the process was essentially “bottom up” with reuse sought at code module. However, more significantly, the development process wasn't oriented towards reuse. The process described in this paper seeks to provide the technology to enable a change where reuse was the norm, and new software functions are only developed where strictly necessary. The technical approach proposed here is founded on the ideas of family analysis [Coplien-et-al98, Mannion-et-al99].

While this work is motivated by the engine control domain, the discussion in this paper is made as general as possible to make it easier to see how to migrate the ideas to other application areas.

Overview

The rest of the paper presents a more detailed description of the process we have developed and illustrates it using examples of enabling technology.

The overall process, including the family analysis method developed for the project, is outlined before each phase of the software development process is considered in more detail. The requirements section describes the requirements process. The architectural style adopted is discussed in the section on software architecture while the specification section discusses the specification notations used within this architectural style. Support for the software architecture in delivered code is discussed in the implementation section. Approaches to testing and test automation are discussed in the section on testing. The evaluation section presents preliminary findings from a demonstrator project. The final section draws conclusions, and discusses the additional work believed to be necessary to develop these research results into a useable industrial process.

Process Description

Family-Based Software Development

Family analysis is concerned with understanding the properties of a family of related systems, rather than individual systems. Families of programs have been the subject of such analysis since early work on software structuring [Parnas76]. In this early work, members of the family were identified and differentiated between by means of their properties. In the case of families of software systems, these properties are

termed features. Features of a family of systems fulfil two purposes: they identify those systems as being members of that family, and they identify individual systems as being different from one another.

Figure 1 gives an overview of the relationships between some of the processes used in family-based development using the RAD notation [Ould95].

The construction processes are organised around a family capability that provides controlled access to the assets needed to build and validate a family member, such as specifications, architectures, components and test cases. The overview shows three interactions:

- 1. an analysis process that builds the capability structure
- 2. an asset development process that creates reusable assets; and
- 3. a project process, which uses the capability structure to retrieve assets.

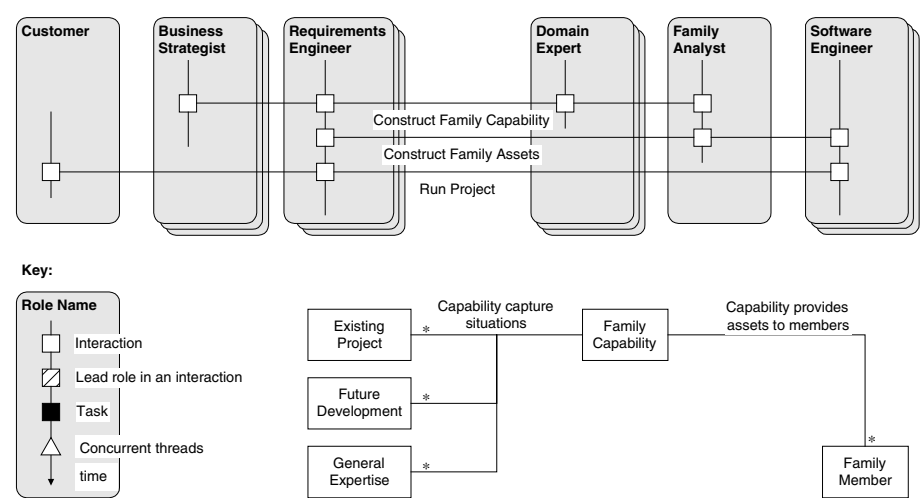


Fig. 1. Family Based development overview

Family Analysis

The initial phase of any asset-based process is to build the asset infrastructure - here termed the “family capability” - whose construction is detailed in Figure 2. The stages of construction are modelled after those found in FAST process [Coplien-et-al98] and Napier's MRAM requirements family work [Mannion-et-al99]. The concepts and structure are largely influenced by family-oriented processes such as MBSE [SEI97].

Scope

A family capability can only fabricate a limited number of different systems, be they under development, or planned for the near future. The scope of a family analysis identifies which systems are under consideration and serves the following purposes:

- It determines which systems - and hence which features - should, and should not, be considered during the analysis.
- It explicitly identifies the system boundary.

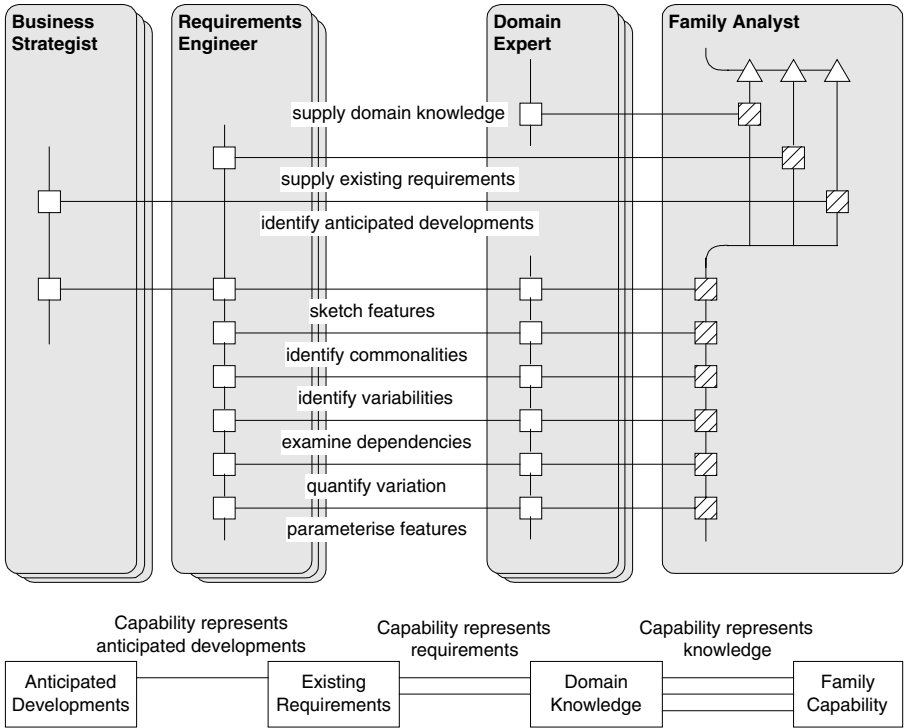


Fig. 2. Family capability construction

The analysis stage is responsible for collating information from within the scope, and organising it to reflect the patterns of features within the family.

For each source of information within the scope, a graph is constructed where nodes are features required by that source and the arcs represent dependencies among those features e.g. mutual exclusion, dependency etc.

A system of commonality operators is used to select to a particular family member from family features. The operators used here are:

- **Common (CO)** - A feature present in all family members.

- **Variable (VO)** - A feature present in only some family members.
- **Parameter (PO)** - A choice that parameterises a feature.
- **Choice (VCL)** - A set of features of which exactly one is selected for any given family member.
- **Selection (VSL)** - A set of features of which at least one is selected for any given family member.

Iteration

Just as the development of a single family member goes through a number of stages of refinement, so too does a family analysis. Features are added at each stage and related to those from the previous stage, most often by noting under which selections the new features are applicable.

Capability Evolution

A family asset base is first constructed with reference to a fixed scope of projects and anticipated developments, which represents the most likely features needed in new projects. However, this cannot be guaranteed to remain applicable so facilities must be provided to handle evolution. In the family analysis scheme, the major influencing factors on the evolution of the system are:

- **Direct Scoping** - In the new scope, some features may be common that previously were not, and others may become variable when they were previously common.
- **Standards** - A change in safety certification standards may call for a process to be performed, or a claim to be made, which cannot be supported because of the process or the technology used in fabricating family members.
- **Technology** - A change in the technology available for the implementation of the family will have an impact on the family capability. The change might be a new technology, such as an advance in automatic code generation techniques, or the obsolescence of old hardware, such as a processor or a family of sensors.

Project Operation

The operation of a project that has access to a family capability is similar to that without; software engineers create artefacts in response to requirements. With access to a family capability, however, the software engineer uses the requirements to select among existing assets, where possible. While family assets will provide for a large proportion of a product, some additional work will invariably need to be performed before that product is complete.

The following are some of the significant aspects of the member project process:

- **Requirements Analysis** - In a family-based project, the requirements represent the features and constraints demanded by a particular customer. With access to a family capability, it is possible to elicit requirements by presenting the current

range of features supported in that capability, and customising the requirements from that point.

- **Fabrication** - A set of requirements *discriminate* between the various features in the family and may call for features that are not present in the current family asset base. The family capability provides a process by which this discriminator is applied to the family assets to fabricate a family member. In the software domain, the most applicable automatable fabrication technologies are generational and component-based reuse.
- **Completion** - alters the product that was produced from the family capability through the provision of missing features.

Project Evolution

A family-based software development project evolves in a similar way to any other software development project, by adding detail and responding to changes in technology, requirements and constraints. In order to provide feedback into the family capability development process, a family-based project has a number of key differences:

- requirements changes can inadvertently alter the scope of the family capability;
- the fabrication process itself may be at fault, requiring a change in the family capability rather than in the project
- the results of completion work must be made available for potential inclusion into the family capability.

The remainder of this paper will describe processes and techniques for the development of a single family member, assuming the presence of a central family process.

Requirements

Previous work in the UTC resulted in the development of the ConCERT (Controlled and Consistent Expression of Requirements using Tables) requirements method [Vickers-et-al96a, Vickers-et-al96b]. This method provided the company with a framework for organising information, a notation for expressing the requirements within that framework and a lexicon for defining the vocabulary of the information recorded in the notation.

Though the ConCERT approach embodied a number of sound principles it did not emphasise the traceability necessary for a family-based software development process. DOORS, a commercially available requirements management tool, has been used to provide the essential traceability needed to satisfy the demands of the safety-critical development process and, in addition, provide the opportunity for tactical reuse.

In comparison with current document-centric requirements processes, the rewards offered by this approach to reuse seem high. The effort expended in document authoring is reduced and the authors are able to draw on a vast quantity of previous

requirements, much like a database. However, the automation of requirements management falls short of tackling the fundamental issues at the root of requirements process improvement and it does not necessarily support the required infrastructure needed for family-based development.

In order to exploit the potential for reuse that exists across development projects a model-based process for the elicitation, negotiation, validation, management and reuse of requirements has been developed which comprises 3 model views:

- **Conceptual** - models record information about the static structure of the entities that comprise the system and its environment (insofar as this affects the definition of system requirements). These static descriptions of the system are the foundation upon which the other model views are built.
- **Functional** - models capture the purpose of the system, or system goals. The use of use cases to represent functional models enables the explicit recording of interactions between the system and external entities described in the static view.
- **Behavioural** - models use state diagrams and scenarios to record the behaviour of external entities and to specify intended system behaviour. In many instances, the control system is responsible for the preservation of correct behaviour in external systems and behavioural models of these systems is critical for the determination of controller requirements.

The process fully utilises domain knowledge in order to develop models of system function, behaviour and design that reflect the inherent properties of the delivered product at every level, from integrated control system through to software. It is the models themselves that provide the necessary information from which to develop requirements specifications. Documentation (traditionally the single medium for requirements process visibility) becomes a view on the models, which are the configured artefacts of the process. The models populate the family capability and selection for family member development is based on the features that they encapsulate.

Software Architecture

Existing architectural styles for dependable systems (e.g. HRT-HOOD [Burns-et-al94, Burns-et-al95], DORIS [Simpson96], and on MASCOT [Simpson86]) are geared towards constraints imposed by the need to collect certification evidence for assessing the implementation of a software system (e.g. by information flow analysis [Bergeretti-et-al85], or schedulability analysis [Sha-et-al90]). Each approach offers several advantages for the implementation of a dependable system, but does not explicitly address the issues of change management and reuse required for family-based development.

The HADES architectural style has been developed in an attempt to address these issues. It is preferred over similar styles for dependable systems because of its emphasis on partial specification rather than functional refinement. This allows the designer to describe localized commitments one at a time, and to later check that the software is still architecturally and functionally consistent. This has benefits not only for rapid prototyping, but also for reuse, as checking for the correct reuse of a

component involves the same checking as for a new component. Similarly, the partial specifications may be ordered to suit the volatility of those specifications, improving the response of the software to change. Figure 3 shows an example HADES diagram for the control software design of thrust reverser.

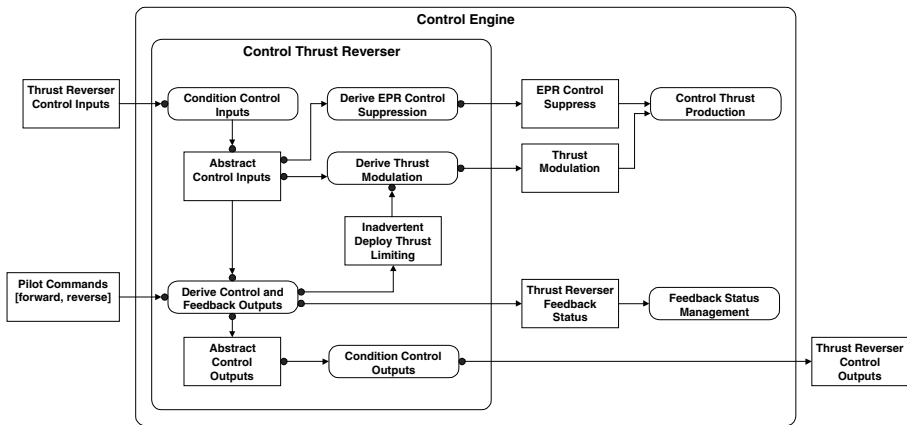


Fig. 3. HADES diagram for a thrust reversal system

In HADES, there are two types of component:

- **Activity** - representing software performing computation. It is represented as a labelled box with rounded corners. The computation may be actively scheduled, or triggered by data flow.
- **Data Area** - representing the information itself. It is represented as a labelled box with square corners, and may include a type identifier.

Components may be decomposed hierarchically. Activities that are decomposed are called coordination components and are responsible for organising the (sub)activities contained within. Activities that are not decomposed are called *terminal components* and contain simple specifications of their behaviour.

Communication between components is specified with a synchronising data flow connector, which may connect one activity to one data area, or one data area to one activity. Connectors have optional attributes to describe the following:

- **Initiation** - Either the source, or the target of the connector is responsible for initiating the connection. A data area will source a connection when its data has been supplied and request a connection if its data is requested. An activity will

source a connection when it has finished processing and request a connection when it begins processing¹

- **Termination** - Connections may be terminated synchronously or asynchronously. In a synchronous supply, the connection is held until the destination has finished with the connection. In a synchronous request, the connection is held until the source is ready to supply. In an asynchronous supply, the connection is held until the supply is complete. Asynchronous requests are available, which make the request but do not wait for it to be fulfilled; in this case, there should also be a matching asynchronous supply.

Specification & Verification

Component Specifications

Coordination and terminal components in the software architecture carry specifications. We have not yet addressed the use of formal specifications to describe the behaviour of a coordination component; behaviour is given by synchronisations between subcomponents. Terminal components, however, have complete specifications expressed using the PFS (Practical Formal Specification) notations [Galloway98, McDermid-et-al98].

The PFS notation has two categories of component: reactive and state-based. Reactive components are stateless functional transformations of their inputs consisting of pairs of guard expressions and definitions. The guards partition the input space of the function. When a guard expression evaluates to true, the corresponding definition is used to calculate the value of the component's output.

State-based components are hierarchical state machines, similar to Statecharts [Harel-et-al88]. PFS state-based specifications differ from traditional statecharts in a number of ways, primarily:

- Concurrent states are not modelled - the architecture is relied on for concurrency.
- Assumptions govern each state, defining constraints on the input values used in any definitions within or transitions from that state.
- Outputs are defined within states rather than on the transitions.

PFS components also carry explicit assumptions. The assumptions specify particular properties of the context in which the component is to be used. When components are composed, each must guarantee the assumptions of the next. The composed component may, in turn, have its own assumptions. The composition of all components thus propagates an assumption out into the environment in which the software operates. An argument must then be formed to show that the environment guarantees the software specification's assumptions.

¹ This is a closed synchronisation model. Activities collect all inputs, then process, then produce all outputs.

Specification Validation

Completeness and determinism properties of the PFS specifications are determined by translating them into formal specifications expressed in a subset of the formal notation Z [Spivey92] and analysing them using an in-house tool [Toyn96, Toyn99]. Included in the generated Z specifications are the accompanying proof conjectures to discharge the completeness and determinism checks, along with English language comments and traceability tags to aid review of the generated formal specification. These annotations also formed the basis of the test descriptions (see below). Coupled with an automated proof capability [Toyn96, Toyn99], this allows the formal analysis to be performed as a development aid rather than a separate post development activity.

Proof tactics were developed for discharging the completeness and determinism conjectures. The consistent structure of the Z specifications (resulting from the automatic generation) allowed the proof tactics to be reused over a number of conjectures. Proof tactics allow approximately 90% of the completeness and determinism conjectures to be discharged automatically. Proof failures result in the generation of a counter-example. This information can be fed back into the family capability much earlier in the lifecycle than using traditional approaches. The high level of automation ensures that the only additional work required is that of locating the errors in the specification based on the counter-examples.

Implementation

It is important to segregate the software architecture from its implementation. The software architecture states *what* boundaries and information flows exist between components but not *how* they are implemented in software. This separation allows freedom in the process to choose an implementation strategy which best satisfies those requirements that lie outside the scope of the software architecture, for example:

- ease of static analysis;
- testability;
- interaction with hardware; and
- space/time efficiency

The software must maintain both the abstract state of the system and the concrete state as observed at the hardware interface and provide a mechanism for relating these states. This relation can be quite complex, for example, when controlling a gas turbine the abstract state is concerned with thrust. This cannot be observed directly, so thrust is inferred from knowledge of pressure ratios in the turbine.

Embedded software functionality is managed by a three-layer implementation architecture where the layers are categorised as follows:

- **Application** - which maintains the abstract state of the embedding system under control in terms of concepts within that system;
- **Translation & Validation** - which maps the physical state of the system observed at the hardware interface to the abstract state of the engine in terms of application layer concepts; and

- **Hardware Abstraction** - which isolates the software from the underlying hardware used to control the embedding system.

Layering allows implementation details in a given layer to be hidden from the layer using the facilities provided. This reduces unnecessary visibility in the software as a whole and helps to control the impact of both software and hardware changes.

Each sub-system in the software architecture represents a vertical slice through the application and translation & validation layers. Each component of the sub-system is allocated to either the application layer or the translation & validation layer² using data flow analysis of component interactions in the HADES diagrams.

Co-ordination components are implemented in a call-return framework with the component simply sequencing sub-activities. Data flow relationships between components are used to generate calling schemes. Sequencing of computations within a co-ordination component is done by topologically sorting the sub-components (both activities and data areas) of a given co-ordination component and sequencing computations according to ordering given by the sort.

State-based components are implemented as procedures whose input parameters are data for evaluating guards and whose output parameters are actions. The state is held within the closest enclosing static run-time scope (since this must remain static over multiple invocations of the procedure) with the set of allowable states represented by an enumeration type.

Purely functional behaviour is implemented as functions whose inputs are data involved in guard evaluation and whose outputs are the set of possible PFS definitions in the specification.

In the early life of the family capability the system provides generative reuse from specifications. As the family capability evolves, the ability to track changes in architecture and specification and relate them in a systematic way to code allows more component-based reuse.

Specification-Based Testing

Testing activities were focused towards specification-based module testing. Module testing presently consumes a large proportion of V&V related costs for safety-critical software and can often add little to the process in terms of number of functional errors found. Achieving structural code coverage of the Module Under Test (MUT) is, however, necessary for certification.

The automatically generated formal specifications were used to automatically generate module tests, which were run using a test harness capable of capturing structural coverage metrics. In combination with mutation testing techniques [Budd81, DeMillo-et-al78], the structural coverage was then used to assess the effectiveness of the automatically generated tests. The high level of automation allowed more time to be spent assessing and optimising the effectiveness of the testing criteria.

² The implementation of a sub-system can therefore be loosely coupled across the upper two layers in the architecture.

Test Generation

The formal specifications representing the reactive and state-based components were first analysed to extract information to guide the testing process. Two methods were used to generate test cases from Z specifications:

- **Partition testing:** The input space of the specification is partitioned into disjoint domains based on information contained within either the types or predicates used in the specification.
- **Fault-based testing:** Specific faults are hypothesised that reveal themselves as mutations of the specification.

The CADiZ theorem proving assistant [Toyn96, Toyn99] was used to automate the process of generating the test cases for particular specifications based on generic formal definitions of testing criteria (e.g. coverage of input space). The proofs instantiated the criteria with selected operands from the specification to form the test cases [Burton-et-al00]. These proof tactics were reused for many different criteria.

The test cases represent constraints on the input parameters to the MUT. These constraints were solved using CADiZ's built in constraint solvers [Berzin99, Clark-et-al97] to generate test data. The test data was then used to automatically populate test scripts.

Evaluating the Tests

The MUT is instrumented for structural code coverage and the metrics are recorded while running the tests. The coverage metrics were used to identify unexercised portions of the code. The untested areas of code were examined and assessed as to whether they were:

- valid refinements of the specification;
- erroneous behaviour; or
- shortcomings in the test cases.

Full code coverage does not guarantee that all faults have been detected. To further evaluate the effectiveness of the tests, mutation analysis [Budd81, DeMillo-et-al78] was performed. Many variations of the program under test were generated where each variation represents a single fault representative of the type a developer may make. The test set is then run against all mutants of the program. The proportion (mutation adequacy) of these mutants that are detected (killed) by the test set is then used to make an assumption about the effectiveness of the test set at detecting all possible errors.

Testing Software under Change

The approach to module testing described above has allowed the amount of work needed to test changes to the software to be minimised. The amount of rework depends on the scope of the change and can be summarised as follows:

- **Interface to module unchanged — specification unchanged:** the test scripts for that component specification can be directly reused to test the updated module.

- **Interface to module changed — specification unchanged:** the abstract test cases generated from the formal specification can be reused.
- **Specification changed:** the abstract and concrete test cases must be generated based on the formal representation of the new component specification.

Evaluation

It is difficult to demonstrate processes and to show that they are effective. However it was felt essential to have some evidence that the proposed new process could produce project benefits. A demonstrator project was undertaken with the objective of showing that the new process could handle initial development and subsequent changes more efficiently and effectively than the existing process.

The demonstrator is based on the software for an existing electronic engine controller (EEC). In particular attention was focused on the control of the thrust reverser mechanism. In essence the functional requirement is to deploy the thrust reverser mechanism, on command from the pilot, in order to decelerate the aircraft during landing, or to achieve “push back” from the stand in an airport.

In principle, all thrust reversers operate in the same way, by reversing the exhaust gas stream using mechanically controlled doors. However, changes in type of doors, for example how they are actuated and locked (to prevent inadvertent deployment during flight) have a significant effect on the control software.

Family-wide requirements were identified from existing requirements documents for small engine controllers. This process was only performed for those aspects of control directly related to the thrust reverser.

Selection among the family requirements gave rise to an instance of a single family member (corresponding to a current small engine thrust reverser) from which hierarchical architectural specifications in HADES were produced.

Software components were specified in PFS reactive and state based notations and the specifications were validated. Implementations were produced in SPARK Ada [Barnes97] using the implementation strategy discussed previously. These implementations were then unit tested.

Table 1 shows the time consumed by each process activity.

Table 1. Man-hours per process activity

Process activity	Man-hours
Identification of family-wide requirements	60
Architectural specification in HADES	30
Software component specifications & validation	80
Software implementation	30
Unit testing	15
Total time	215

Table 2. Man-hours per process activity - thrust reverser component

Process activity	Man-hours	
	Initial	Change
Software component specifications & validation	6	4
Software implementation	2	1
Unit testing	3	2
Total time	11	7

The specification was subject to a change reflecting the introduction of a different thrust reverser door locking policy, specifically the introduction of locks that were under software control rather than hardware control. Table 2 shows the time taken for the development of the initial locking policy and the subsequent change. The four hours for software component specifications consists of two hours of negotiation and two hours to describe the impact of the change and produce the specifications. The software implementation required only changes to the translation and validation layer and the addition of two hardware ports to the hardware abstraction layer (representing the lock control interfaces). As the implementation interface remained the constant between changes, but the specification had changed, test cases had to be regenerated. The reduction in time arises because of the known conversions between specification and implementation types derived for the first implementation.

The original engine controller on which this demonstration was performed was not developed as part of a family and did not have a target architectural style. It is therefore difficult to perform a comparison of the time taken to make this change under the new process and in the original engine controller. It is clear, however, that introducing such a change in the current engineering process would have been significantly more expensive.

There is an up front cost associated with the family analysis and production of the target software architecture, however, once this is in place, the later stages of the process can be performed rapidly, especially in the presence of change.

Conclusions

The work described here has shown the feasibility of a family based development process for engine controllers. The work on family analysis has extended previously published ideas [Coplien-et-al98] in ways that facilitate family-based development. In particular, the refinement of the ideas of variability, and the introduction of dependency modelling can be used to guide development towards effective and systematic reuse.

A considerable attraction of the approach is that it brings many of the benefits of formal methods to projects, but in such a way that they can readily be adopted in an industrial software development process. Our approach to software architecture is less general, but should be applicable to a range of real-time embedded systems. As well as supporting the family-based development process it has benefits for dealing with hardware obsolescence and component change - problems that beset most

embedded control systems. The testing work has been more generally applicable and has been adopted within the company.

Future Work

The demonstration presented in this paper deals with state-based code, but there is a need to show the capability of the process to deal with the other types of code, especially PID (Proportional, Integral, Derivative) control loops, as this is the “core” part of the controller.

Little attention has been paid to the safety process. In principle it seems possible to adopt a family approach to the safety process. However safety is a “holistic” property, and it is far from clear if this can be handled in the “modular” fashion necessary to support family analysis. This remains a major research challenge, which we are only just beginning to study.

There are many opportunities for improvement in the enabling technologies including, but not limited to: requirements for hardware/software co-design, architecture verification and automation of the code generation process. We have also prototyped a web-based process management tool to automate much of the process management.

Summary

In summary we believe we have made some important steps in developing technology for building controller software using a family-based process. We are optimistic that we can build on these results to facilitate a business process change for Rolls-Royce and that many of the results will transfer to other domains.

Acknowledgements

The work described herein would not have been possible without widespread support in York and at Rolls-Royce. In particular we wish to thank Paul Emberson, Andy Galloway, Jamie Hodgkinson, Andrew Payne, George Taylor and Eddie Williams. We gratefully acknowledge the financial support from Rolls-Royce and the EPSRC CONVERSE Grant (GR/L42872).

References

- [Barnes97] John Barnes. High Integrity Ada The SPARK Approach. Addison-Wesley Longman Ltd, 1997.
- [Berzin99] Sergey Berzin. The SMV web site.
<http://www.cs.cmu.edu/~modelcheck/smv.html>, 1999. The latest version of SMV and its documentation may be downloaded from this site.

- [Bergeretti-et-al85] J.-F.Bergeretti and B.A.Carré. Information-Flow and Data-Flow of while-Programs. *ACM Transactions on Programming Languages and Systems*, 7(1):37-61, January 1985.
- [Budd81] Timothy A.Budd. Mutation analysis: Ideas, examples, problems and prospects. *Computer Languages Program Testing*, 10(1):63-73, 1985.
- [Burns-et-al94] A.Burns and A.J.Wellings. HRT-HOOD: A structured design method for hard real time systems. *Real Time Systems Journal*, 6(1):73-114, January 1994.
- [Burns-et-al95] A.Burns and A.J.Wellings. HRT-HOOD: A Structured Design Method for Hard Real Time Ada Systems. Elsevier, 1995.
- [Burton-et-al00] Simon Burton, John Clark, and John McDermid. Testing, proof and automation, an integrated approach. In *Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification*, June 2000.
- [Clark-et-al97] John Clark and Nigel Tracey. Solving constraints in LAW. LAW/D5.1.1(E), European Commission - DG III Industry, 1997. Legacy Assessment Worbench Feasibility Assessment.
- [Coplien-et-al98] James Coplien, Daniel Hoffmann, and David Weiss. Commonality and variability in software engineering. *IEEE Software*, pages 37-45, November/December 1998.
- [DeMillo-et-al78] R.DeMillo, R.Lipton, and F.Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11:34-41, 1978.
- [Galloway-et-al-98] Andy Galloway, Trevor Cockram, and John McDermid. Experiences with the application of discrete formal methods to the development of engine control software. *Proceedings of DCCS '98. IFAC*, 1998.
- [Harel-et-al88] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michael Politi, Rivi Sherman, Aharon Shtull-Truaring, and Mark Trakhenbrot. STATEMATE, a working environment for the development of complex reactive systems. *IEEE Transactions On Software Engineering*, 16:403-414, 1988.
- [Hill-et-al94] J.V.Hill, J.A.McDermid, R.Rimmer, and B.R.Whittle. Re-use of engine control technology. 1994 Avionics Conference and Exhibition, ERA Technology, 1994.
- [Mannion-et-al99] Mike Mannion, Barry Keepence, Hermann Kaindl, et. al. Reusing Single System Requirements from Application Family Requirements. In *Proceedings of the 21st International Conference of Software Engineering*, pages 453-463, May 1999.
- [McDermid-et-al98] John McDermid, Andy Galloway, Simon Burton, John Clark, Ian Toyn, Nigel Tracey and Samuel Valentine. Towards industrially applicable formal methods: Three small steps, one giant leap. *Proceedings of the International Conference on Formal Engineering Methods*, October 1998.
- [Neighbors84] J Neighbors. The Draco Approach to Constructing Software from Reusable Components. *IEEE Transactions on Software Engineering*, SE-10:564-573, September 1984.
- [Ould95] M.A.Ould. Business Processes: Modelling and analysis for re-engineering and improvement. Wiley, 1995.
- [Parnas76] D.L.Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, 2(1):1-9, March 1976.
- [Prieto-Díaz90] R Prieto-Díaz. Domain Analysis: an Introduction. *Software Engineering Notes*, 15(2):47-54, April 1990.
- [Sha-et-al90] L.Sha, R.Rajkumar and J.P.Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 9(9):1175-85, September 1990.
- [Simpson86] H.Simpson. The MASCOT method. *Software Engineering Journal*, 1(3):103-20, May 1986.
- [simpson96] H.R.Simpson. Layered architecture(s): Principles and practice in concurrent distributed systems. In *IEEE Symposium on Parallel and Distributed Processing*, 1996.

- [SEI97] Software Engineering Institute. Model-Based Software Engineering. <http://www.sei.cmu.edu/mbse/>, 1997.
- [Spivey92] J.M. Spivey. The Z Notation - A Reference Manual. Prentice-Hall, Second edition, 1992.
- [Toyn96] Ian Toyn. Formal reasoning in the Z notation using CADiZ. 2nd International Workshop on User Interface Design for Theorem Proving Systems, July 1996.
- [Toyn99] Ian Toyn. The CADiZ web site. <http://www.cs.york.ac.uk/~ian/cadiz/>, 1999. The latest version of CADiZ and its documentation may be downloaded from this site.
- [Vickers-et-al96a] A.Vickers, P.Tongue and J.Smith. Complexity and its Management in Requirements Engineering. INCOSE UK Annual Symposium – Getting to Grips with Complexity, Coventry, UK, 1996.
- [Vickers-et-al96b] A.J.Vickers, J.E.Smith, P.J.Tongue and W.Lam. The ConCERT Approach to Requirements Specification – Version 2.0. Technical Report YUTC/TR/96.1 Rolls-Royce University Technology Centre, University of York, York, UK, YO10 5DD, November 1996.

Enabling Local SPI in a Multi-national Company

Peter Fröhlich¹, Horst Lichter², and Manfred Zeller¹

¹ ABB Corporate Research Center Heidelberg
{peter.froehlich, manfred.zeller}@de.abb.com

² Department of Computer Science, RWTH Aachen
lichter@informatik.rwth-aachen.de

Abstract. ABB has a long tradition of improving software processes and of applying CMM in SPI projects. This paper presents the structure and results of a company-wide SPI initiative called ASPI. The initiative aims to complement the existing local SPI projects by harmonizing processes, methods and tools to be used in R&D software development throughout ABB in a common framework. We give an overview of the structure of this framework and present in detail the common decision model for product development projects, which forms an important part of the framework. This model is generic and has to be tailored to the needs of local ABB R&D units. We conclude by discussing the experiences gained during introducing the decision model and give a brief outlook on the SPI activities planned in the future.

1 Introduction

Many companies in the software business have realized that substantial gains in the productivity of software development and in the quality of software products can only be achieved by improving the software process. Such improvements have turned out to be costly and bearing a high risk of failure, since the software process is a complex system of relationships among processes, technologies and people.

Results and experiences of industrial software process improvement (SPI) initiatives have been recently published. These initiatives have typically been part of large, company-wide process improvement programs. For instance Wohlwend [13] describes Schlumberger's and Komiyama [6] presents NEC's SPI program.

ABB is a group of companies that operate in different countries and business areas. The companies are managed locally. As a consequence, smaller software process improvement initiatives have emerged independently at different ABB companies, with specific improvement objectives. The results and experiences of these improvement activities are discussed in detail in Lichter [8] and Welsch [12].

In this paper we present the structure and results of a company-wide SPI initiative called ASPI (ABB Software Process Initiative). This initiative is funded by the ABB research program "Industrial IT Architecture and Software Processes" and is carried out by consultants from ABB Corporate Research. The initiative aims at defining and setting up global software process elements and coordinating local SPI efforts.

The paper is organized as follows. Chapter 2 presents all elements of ASPI. Chapter 3 describes in detail the project control model, which is the core element of ASPI. In chapter 4 we introduce the ABB Gate Model. This model is part of the project control model and serves as an overall decision model. Chapter 5 presents how the Gate Model is introduced and implemented in local ABB R&D units. Chapter 6 summarizes our conclusions and gives an outlook on SPI activities planned in the future.

2 Overview of ASPI

To complement the existing local SPI activities, and gather global synergies (see Thomas [11]), ABB started in 1999 its company-wide SPI initiative ASPI. ASPI stresses business and project management issues and tries to link SPI activities to business goals (see Debou [3]).

In particular, ASPI has been working intensively towards two goals:

1. Harmonize processes, methods and tools to be used in R&D software development throughout ABB in a common framework.
2. Create a culture of continuous self-improvement in the R&D software development units with the goal to work on an efficient and mature process level.

2.1 Organization

ASPI was started as a research project by ABB's Corporate Research organization. To establish SPI as a permanent function in ABB, a unit in ABB's group-wide process organization has meanwhile been created to own the SPI activities. As shown in Figure 1, several organizations contribute to ASPI:

- **ABB Group Processes:** The ABB group has a central process organization called Group Processes, who owns all common ABB processes, e.g. for product development or business project execution. A unit in this organization has been established to centrally coordinate the different SPI projects. This unit initiates the local SPI activities by addressing local management and monitors the status of the SPI activities. It further ensures consistent application of the standards developed within ASPI.
- **Corporate Research:** After setting up the initiative, Corporate Research evaluates, packages, and transfers knowledge from external partners (universities, consultancy companies) to the process organization. Experts from Corporate Research further support business units in SPI projects.
- **R&D Units:** Selected R&D units are required to budget for SPI, create SPI plans, conduct improvement activities and report the status of the activities to the process organization. In return part of the external cost of the unit's SPI activities is covered by corporate fund of the ABB group.

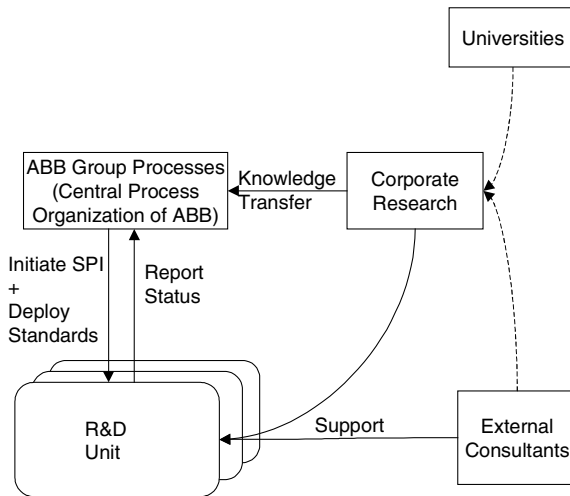


Fig. 1. Organization of ASPI

2.2 A Common Framework for Software Development and Maturity

ASPI approaches the R&D units from both top-down through the ABB-wide common framework and bottom-up by supporting local SPI projects, as indicated in Figure 2.

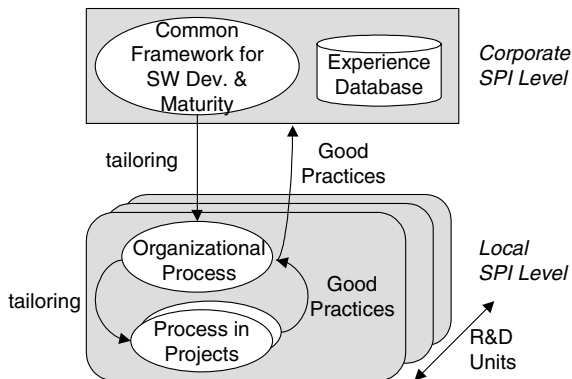


Fig. 2. Company and local SPI levels

On the top-down side ASPI is providing a Common Framework for Software Development and Maturity, consisting of the following elements:

- A project control model: This model defines a common language and common procedures in the areas Project Decision Model, Project Management Model, and Development Model. It is covered in more detail in section 3.
- A model for execution of process improvement projects: This model is called the Improvement Engine. It is in spirit similar to the SEI Ideal model [9] and defines the phases of a CMM-based improvement cycle.
- A set of 4 common metrics: The Common Framework defines metrics for performance baselining, covering time, effort, quality and functionality.

The Common Framework establishes a common language throughout the company and provides common management procedures. Furthermore, it generates management commitment to SPI activities by providing a set of tools to support planning and tracking of such activities.

2.3 Experience Database

Another key element of ASPI is the experience database (EDB). The goal of the EDB is to facilitate sharing and reuse of experience. It therefore has a similar motivation as the experience factory [1]. However, ASPI had to construct the EDB with minimum administrative overhead and the goal of short payback time. Thus, we are targeting a lightweight approach to experience reuse, comparable to the environment described by Houdek [5].

In the EDB, experiences are represented using a structured template. Content is controlled by a small organization, called the experience control board (ECB). There are 4 levels of experiences with different requirements on elaboration and degree of reuse of the experience, as sketched in Figure 3. The levels are

- Entry level experiences: This is the most informal type of experience, e.g. an observation made in one R&D unit. It must contain contact information, experience description and lessons learnt. An example for this category is the description of a daily build process of one local R&D unit.
- Experiences: In this category a certain level of reuse is required. Based on the application of the experience success factors have to be given as part of the experience package. An example for this category is a code review seminar conducted and analyzed at different R&D units.
- Good Practices: On the next level, a proven experience can become a good practice. Good practices must contain a cost-benefit analysis and a guide for introducing the practice. The experience package must contain all necessary artifacts for applying the practice. An example for this category is a test process used consistently in different R&D units of a business area.
- Approved Policies: A good practice can become a mandatory process element upon decision by the process organization. An example for this category is the ABB Gate Model.

The classification of the experience into one of the levels is decided by the ECB after review.

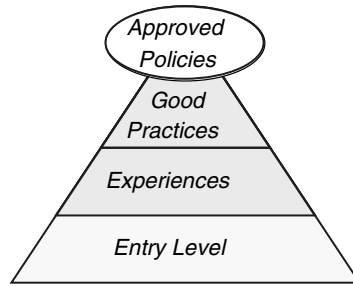


Fig. 3. Levels of experience packages

The EDB is accessible for every ABB employee through a portal, which contains the common framework, the experiences, discussion groups, announcements, etc. The main intended users are however consultants and change agents. These users apply the experiences, when supporting SPI projects in the R&D units. They are able to select appropriate experiences based on the context factors in the experience packages.

The challenges in sustaining the EDB are (1) to keep the users interested in the EDB and (2) to ensure that new experiences are constantly added. The integration in a web site helps achieve (1). ASPI publishes additions to the Common Framework, announcements and edited articles regularly on the EDB web site. Challenge (2) is harder to address, since the ECB is too small to edit all experiences and the R&D units themselves need incentives for creating them. We solve this through the funding scheme of SPI projects. In the application for the corporate funding share, the consultant / change agent has to identify a candidate for an experience out of the project. At the end of the project, an experience has to be delivered to the ECB.

2.4 CMM-Based SPI Activities

The top-down approach is complemented by bottom up SPI activities within the different R&D units. ABB has a long tradition of the application of CMM in such SPI projects (see Welsch [12]). Over the years ABB has developed a toolbox of several types of CMM assessments:

- CBA IPI [4]: This „official“ CMU-SEI assessment type is applied by larger ABB R&D units, who have been investing in SPI for a longer period. The effort of this method is justified by its scientific accuracy. ABB works with CMU-SEI to educate internal assessors for this method.
- ABB Mini CMM Assessment: The first ABB Mini CMM Assessment method was created in 1993 based on the study of the CMM [10] and has been revised based on our experiences with the CBA IPI. It tries to generate comparable results with fewer assessors and less effort. While the redundancy in the assessment team (2 to 4 assessors) is not as high as in the CBA IPI, we maintain consistent results by strict requirements on education and experience of lead assessor and assessment team.

- **CMM Onboard:** Since 2000, ABB is also using the CMM Onboard process developed by Q-Labs AB. CMM Onboard is a process, which allows development projects to reach CMM level 2. In this process a graphical representation of the CMM level 2 in the form of a board is created and constantly updated during the improvement project. We have found this process useful especially for small R&D units, who are at the beginning of their SPI efforts.

Over the last years, restructurings and mergers have impacted many R&D units. While it is difficult to achieve high maturity levels in such a dynamic environment, the knowledge from CMM levels 2 and 3 is seen by most units as critical for establishing a dependable process culture.

3 The Project Control Model

The project control model for product development projects consists of four layers (see Figure 4). The product planning process forms the topmost layer of the model. It serves to systematically plan and manage the product portfolio.



Fig. 4. Layers of the project control model

In order to develop a product and to control the development progress, the project control model defines a Project Decision Model (called Gate Model), a Project Management Model as well as a Development Model.

3.1 Project Decision Model

The ABB Gate Model is the common decision model for product development projects. The ABB Gate Model helps to make the project status visible and provide relevant data as the basis for business decisions. This is achieved through 7 defined decision points (called Gates G0 to G6) during the development project's lifetime. One additional check point (G7) after the project is used for checking the results of

the project and feeding back experiences to the organization. We present the Gate Model in more detail in section 4.

3.2 Project Management Model

The Project Management Model helps the project manager to run a project according to the ABB Gate Model and to provide the information required for the gate decisions. It introduces a common terminology for project management activities throughout ABB and establishes ABB-wide procedures for project steering.

3.3 Development Model

The development model refers to the actual software process of the R&D unit. Due to the great variety of businesses and products in the ABB group, there is not much potential for standardization on this level. However, the Common Framework provides guidance on how software development models with different lifecycles (sequential, incremental, and evolutionary) are used with the ABB Gate Model and the Project Management Model.

4 The ABB Gate Model

Decision models for product development projects have been discussed by Cooper [2]. They have paramount importance for most product-based enterprises, since they

- guarantee product quality and readiness of the different functions in the organization to launch a new product, and
- enable timely business decisions on project continuation or termination based on project status and market criteria.

4.1 Controlling Projects by Gates

A gate is a decision point in a project where those who are responsible for the outcome of the project evaluate the achieved results from a business point of view and determine whether to continue the project or not. A decision to continue may of course include alterations to the project such as changed scope or plan.

At each gate, the status as well as the business opportunities and risks for the project are discussed in a gate meeting. To keep the gate meetings focused on the business decisions, the ABB Gate Model includes a gate assessment before each Gate. Figure 5 depicts the Gate Model in the context of product development projects.

It is important to understand that the Gate Model works on the top level of the product development project. A product development project typically consists of different parallel activities executed by different functions in the organization:

- Marketing
- Sales

- Product Management
- Software Development
- Service
- Training
- Quality Management

In the Gate Model the progress and readiness of all these functions is checked and the feasibility of the project is assessed taking all these functions into account. Thus, the Gate Model is a decision model for the whole product development project, while a software development process is a technical model, which covers only the software development part.

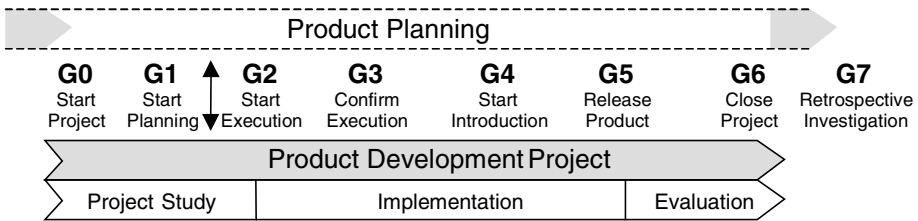


Fig. 5. Gate Model for product development projects

As shown in Figure 5, a product development project is divided into three phases: the project study phase, the implementation phase and the evaluation phase. Gates are used to control the project during these phases.

The gates serve the following purposes:

- Gate 0: Start Project.
Initiates the project study phase. The focus between G0 and G1 is on analysis of the requirements.
- Gate 1: Start Planning.
Defines the scope of the project. The requirements agreed here will control the planning made between G1 and G2.
- Gate 2: Start Execution
Marks the agreement on requirements, concept, and project plan. The focus from G2 to G3 is on specification of functions and architecture.
- Gate 3: Confirm Execution
Confirmation that target dates can be met and that the project executes according to the project description and plan. After G3, the focus is on implementation.
- Gate 4: Start Introduction
Release for acceptance testing. Focus is on validation, on preparation for the market introduction and on production preparations.
- Gate 5: Release Product
Hand-over of the results to the line organization. G5 indicates also that the project activities should be finished and focus in the period to G6 is on finalizing any remaining issues.

- Gate 6: Close Project
The project is terminated.
- Gate 7: Retrospective Investigation
A follow-up of the project to check if the results are satisfactory, and to feed back experiences to the organization.

4.2 Staffing a Gate Oriented Decision Model

In order to apply the Gate Model in development projects, the corresponding activities (e.g. planning the gates, performing gate assessments) have to be carried out. For this end the Gate Model introduces the following roles.

- Project Sponsor
The manager responsible for the development and maintenance for the affected product lines. The project sponsor should have the authority to start and stop the project, i.e. have influence on the product portfolio and the economical power to increase or decrease the funding of the project.
- Gate Assessor
The gate assessor has the overall responsibility to report the status of the project that indicates whether the project is ready to pass a gate or not. The gate assessor is typically from QA, an external assessor or the project sponsor. It is important that the gate assessor is objective.
- Gate Meeting Participants
The gate meeting participants appointed at G0 will have the responsibility to assist the Project Sponsor in evaluation of the project at the gate meetings. The participants at the G1 – G7 meetings should be the responsible managers for development, quality, sales, marketing, service and training as well as the project sponsor, product manager and the project manager. Each participant has the task to monitor the progress in the project with respect to his/her part of the organization.

4.3 The Gate Assessment Process

The ABB Gate Model requires each project to perform gate assessments to prepare gate decisions. The gate assessment reviews the status of the project and prepares the relevant information for the gate meeting. In order to prepare the gate assessment report checklists containing the central aspects to be assessed are used. The aspects covered at each gate include the following business and technical aspects of the project:

- The *benefits aspect* covers the business aspects of the project, e.g. the benefits for the customer, benefits for ABB, profitability of the product, market situation, and competitor analysis.
- The *status aspect* covers the progress of the project (including the different subprojects) compared to plan.
- The *resource aspect* covers the availability of suitable resources for the project.
- The *technology aspect* covers the technical feasibility of the project.

The manager responsible for the affected products (the project sponsor) initiates the gate process in collaboration with the project manager. The first step is to perform an assessment and to prepare the gate decision material. The input to the assessment is documents prepared by the project and gate assessment checklists. The assessment is typically done over an extended period and involves the project manager and a gate assessor. The output from the assessment is signed checklists and an assessment report.

Based on the assessment report, the decision to continue (possibly with changes) or terminate the project is made at the gate meeting, as shown in Figure 6.

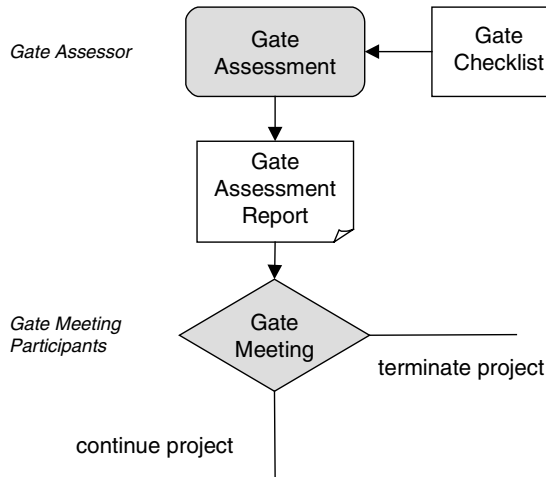


Fig. 6. Gate Assessment Process

5 Implementing the Gate Model

5.1 The Implementation Process

To get the most value of the Gate Model, every R&D unit has to incorporate it into its own processes. To implement the ABB Gate Model there are a number of concrete steps that should be performed. By experience these should be taken in the following order:

1. Sort out the structure in R&D management regarding how decisions are made and map the ABB Gate Model roles to the organization.
2. Decide what project types the ABB Gate Model should be applied to. Decide on a minimum project size, for which it makes sense to apply the Gate Model.
3. Make the gate checklists complete regarding extra checkpoints and document references.

4. Decide how metrics should be collected and reported.
5. Run a pilot project.
6. Decide on how the Gate Model should be rolled out and roll it out.

In order to support the R&D units to introduce and tailor the Gate Model a one-day training course is available. The course is intended for line managers and project managers. It explains the Gate Model and includes several exercises (including two simulated gate meetings) and discussions on how it fits in the software process culture of the R&D unit addressed. Typically, this course forms the starting event in the local gate model implementation process.

5.2 Experiences

Since 1995, ABB's R&D units have been using locally developed decision models with gates. The ABB Gate Model as described above has been piloted since 1999 and officially rolled out throughout the company in March 2000. It is today used by 1200 software developers in ABB's critical R&D units. In these two years, we have discovered the following benefits over earlier ABB models and models from the literature:

- Common language. The mandatory elements of the Gate Model (numbering and names of the Gates, main items in the checklists) establish a common culture in the company. Everyone in the R&D units knows what "G2" means, and which main deliverables have to be prepared for that gate. This commonality enhances cooperation and gives better visibility of the status of critical product development projects.
- Clear separation between decision model and development model. The gate model is the interface between the development project and the product line organization. For the product organization, it is important to understand if they will make profit with the project result and if the progress of all parts of the project is sufficient. The Gate Model provides this information at critical points of the development project without burdening the gate meetings with the intricacies of the software process. This is particularly important for incremental development models, like the Rational Unified Process [7]. In an incremental project, there will be technically motivated internal iterations, which are not visible in the Gate Model. Other iterations will create a product that has to be maintained by the organization. These external deliveries are visible in the gate model, and a Gate 5 is required before the product becomes operational at a customer site.
- Business and quality focus. Early (local) versions of the ABB Gate Model interpreted the gates mostly from a quality assurance perspective and did not explicitly address the business perspective of the project. The gate meetings were often restricted to checking the existence and quality of documents and discussions of technical problems. These deficiencies have motivated the current gate process, which separates assessment from gate meeting, and leads to shorter, more focused gate meetings, and more conscious business-oriented go / no go decisions.

In the Gate Model introduction projects, we have further found the following critical success factors:

- Staffing of the sponsor role. Since the sponsor is ultimately responsible for the business decision, it is important that the sponsor is known and has the competency to make this decision. To identify the sponsor for a project is an important and non-trivial step in a large, matrix-structured company.
- Training Course. The one-day training course has proved essential. We have observed, that units, who have simply downloaded the Gate Model from the EDB without training have suffered from misinterpretations. Without training, the decision making process of the Gate Model can be misunderstood, or it can be mistaken as a sequential development model instead of a decision model.

6 Conclusions, Lessons Learned, and Outlook

In this paper we have presented ABB's approach to improving its software product development processes. While the focus of former SPI initiatives was to improve local processes as well as technological skills, ASPI is a company-wide initiative focusing on product and business driven improvements. Although ASPI is not completed, the following can be summarized:

Because ASPI is a top-level management driven program it is visible throughout the company and has been recognized as an important endeavor. This is in our opinion a prerequisite to develop and implement common processes in a multinational company. Moreover, this initiative has established the central coordination of local SPI activities in the ABB organization, which makes them sustainable.

We can identify two essential success factors of our initiative:

- First, there is a clear separation of common management processes that have to be applied at all units, and local technical processes that are under control of each unit. The common management processes establish a common language throughout the company and create a defined communication channel between line organization and project organization providing a better foundation for business-oriented decision making. The local units maintain the responsibility for their technical processes, including required interfaces to and tailoring of common processes.
- Second, the chosen SPI organization (ASPI, the ABB Group Processes responsible unit and the local SPI organization at the R&D units) guarantees that the common processes are introduced and used, that local SPI projects are conducted and that experiences and good practices are collected and analyzed.

Another important aspect of ASPI is that at first product and business oriented processes have been addressed (by means of the Gate Model). Thereby the product development management was involved right from the start in the process definition. This raises the awareness of the importance of software process improvement and the necessity to invest in SPI activities as well as the acceptance of the resulting common processes.

After the common Gate Model is defined, approved and introduced we are currently working on a common Project Management Model supporting project managers to run a project according to the ABB Gate Model and to provide the

information required for the gate decisions. It introduces a common terminology for project management activities throughout ABB and establishes company-wide procedures for project steering. The Project Management Model will be packaged with a toolbox for project managers, based on good practices from ABB R&D units and external sources.

References

1. Basili, V., G. Caldiera, D. Rombach (1994): The experience factory. In Marciniak (ed.) *Encyclopedia of Software Engineering*, vol 1. John Wiley & Sons, S. 469-476.
2. Cooper, Robert G. (1993): *Winning at New Products: Accelerating the Process from Idea to Launch*, 2nd edition. Perseus Press, 1993.
3. Debou, C., A. Kuntzmann-Combelles (2000): Linking Software Process Improvement to Business Strategies: Experiences from Industry, *Software Process: Improvement and Practice*, No 5, pp 55-64.
4. Dunaway, D. K., and Masters, S. (1996): *CMM-Based Appraisal for Internal Process Improvement (CBA IPI): Method Description*, Technical Report: CMU/SEI-96TR-007, Software Engineering Institute, Pittsburgh, PA, 1996.
5. F. Houdek, K. Schneider and E. Wieser (1998): Establishing experience factories at Daimler-Benz - An experience report. In *Proceedings of the 20th International Conference on Software Engineering (ICSE) 1998*, IEEE Press, Pages 443-447.
6. Komiya, T., T. Sunazuka, S. Koyama (2000): Software Process Assessment and Improvement in NEC – Current Status and Future Direction, *Software Process: Improvement and Practice*, No 5, pp 31-43.
7. Philippe Kruchten (2000): *The Rational Unified Process, An Introduction*. 2nd edition, Addison-Wesley Object Technology Series, 2000.
8. Lichter, H. C. Welsch, M. Zeller (1995): Software Process Improvement at ABB Kraftwerksleittechnik GmbH, In P.Elzer, R.Richter (eds.) *Proceedings of MSP'95 Experiences with the Management of Software Projects*, Karlsruhe, IFAC Conference Proceedings, Elsevier.
9. McFeeley, Robert (1996): *IDEAL: A User's Guide for Software Process Improvement (CMU/SEI-96-HB-001, ADA 305472)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
10. Paulk, M., B. Curtis, M. Chrissies, C. Weber (1993): *Capability Maturity Model for Software*, Version 1.1. Carnegie Mellon University.
11. Thomas, M., F. McGarry (1994): Top-down vs. bottom-up process improvement, *IEEE Software*, 11(4), pp 12-13.
12. Welsch, C., H. Lichter (1997): Software Process Improvement at ABB - Common Issues and Lessons Learnt. *Proceedings of Software Quality Management SQM 97*, Bath UK, March 1997.
13. Wohlwend H., Rosenbaum S. (1994): Schlumberger's Software Improvement Program. *IEEE Transactions on Software Engineering* (20), No. 11, S. 833 - 839.

Project Improvement as Start-Up

Ton Dekkers

IQUIP Informatica B.V., Goudsbloemvallei 50,
5237 MK DEN BOSCH, Netherlands
a.j.e.dekkers@iquip.nl, qtm@interdependent.nl

Abstract. “We boldly go where no man has gone before”. We want to announce new products and services and use matching supporting information systems. All this more rapidly than before and with higher quality. Thinking about time to market, you will come automatically to the question: “How to improve the process to be able to deliver more rapidly?” But the most important of all is not the process; it is the product that matters. Very simple: A product delivered in time that is not working as expected is not a good product. We need to be sure that the product meets the expectations. In this paper we will look in particular in depth to activities needed for defining the product specifications to assure the product will meet the expectations. This we do based on Quality Tailor-Made. Quality Tailor-Made (QTM) is a risk-based approach to improve product as well as process in a project. The project is then a start-up for overall improvement.

1 Introduction

When an information system is developed, the customer's interest is leading. The information system (IS) wanted by the customer is an implementation of the user requirements, the business needs. Moreover the information system should meet not only the requirements specified, but also the implicit requirements and all other expectations. In order to achieve this, the (end-)user has to be aware of the fact that he or she has to provide all necessary information to the developer. On the other hand the developer should be aware that it is difficult to think of all the aspects that could be important for the system. So the challenge of the developer is to help getting the requirements clear and to assure that from the technical point of view the user gets the right solution for his needs. The developer has to keep in mind: the solution is not a highly sophisticated state-of-the-art technical product, but a product fit for purpose that meets all the expectations of the user. Furthermore the product has to fit within the time and money constraints. The user has to be aware to set the scope of the system or the project in such a way that it is possible to develop a system within budget. Here you are, the twofold human-centred software perspective: process control versus product quality. When both goals are achieved we will get customer satisfaction.

2 Customer Satisfaction

A model is used to understand from which customer satisfaction can be assessed. The model expresses the customer satisfaction using four key related components. As shown in the figure, the model components are:

- Business requirements (arisen from new or changed business strategy).
- Quality aspects (the human factor, the expectations).
- Functional design (transformation of the requirements including quality levels, development criteria and constraints).
- Development process (process capability and people competence).

This model represents optimal result if all its components maximise their contribution to customer satisfaction. For example, customer satisfaction is achieved when the product (information system) matches the needs. Only this is not possible when the requirements are not transformed accurately into the design. The design not only reflects the projection of the requirement on the development environment. It also directly influences the development process, the project organisation and competence of team members involved. The right process execution will deliver trustworthy product quality.

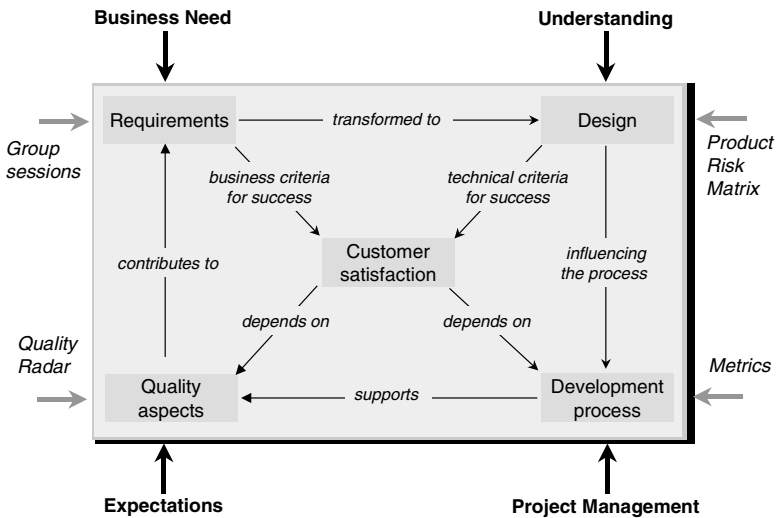


Fig. 1. The Customer Satisfaction Model

The left triangle (Requirements, Customer satisfaction and Quality aspects) reflects the product area. The customer cannot be satisfied when the IS is not functioning in the right manner and is not functioning according to expectations. The right triangle (Design, Customer satisfaction, and Development process) reflects the process area.

The customer will also not be satisfied when it is not produced well; he expects the information system to be delivered in time and within budget. The customer wants value for money.

Software Control has looked for an approach to be able to manage all the aspects mentioned before and during the project. Quality Tailor-Made is developed based on experience in Quality Assurance in a great number of projects. For us, Quality Tailor-Made is the approach to maximise customer satisfaction. QTM is no new development method; QTM does not commit to replace existing tools or standards. No, it is just a practical approach that helps to find out what's important to do and where you must think off in a project. All in relation to the focus on the customer. What do we need to do to satisfy the customer? How can we improve the quality of the product? What is the optimal process to create the product? The message of QTM is: consider system development projects not only from the point of view of time or money, but by choosing an approach which keeps in mind the explicit and implicit requirements of the customer. Achieve goals on both the product and process and control the risks that could put obstacles in the way.

3 Quality Tailor-Made

3.1 Risk Areas

Looking at the customer satisfaction model it is obvious the risks can be split into risks related to the deliverable product and risks related to the process to get to the product. Analysing the triangles in the model we can recognise two kinds of risks in the product area: risks in the functionality and risks related to quality aspects. In the process area two types of risks can be distinguished: risks in relation to internal planning and control of the project organisation internally and those in relation to the outside factors.

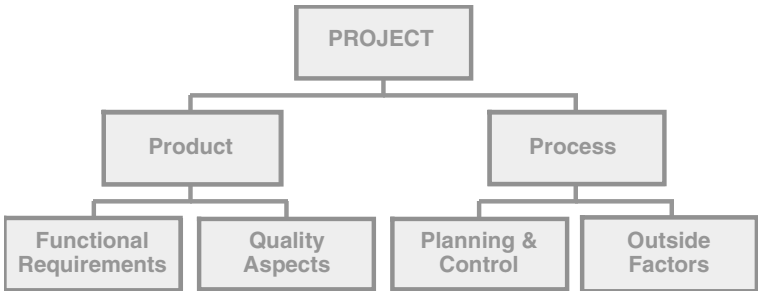


Fig. 2. The Four Risk Areas

Briefly, the risk in respect of the product: the right system is not built. The risk in respect of the process: the system is not built right. To get customer satisfaction you need to take care of both, product and process.

When you analyse the activities of managing a project, you can categorise these activities in five steps. These steps are aimed at eliminating the risks related to the product and risks related to the product development process. These five steps will be considered for all four risk areas: Functional Requirements, Quality Aspects, Planning & Control and Outside factors.

The first step is to *identify*: what do we really want? In this step we will try to identify all possible aspects within a risk area. When we *value* the objectives, a selection will be made from all aspects that are relevant for this specific project. There is always a hierarchy in the properties. From properties that are considered to be important, the requirements must be *specified* in detail. For instance, if response time of the system is important, criteria for response time must be written down, and quantitative aspects like the percentage tolerance must be set.

There are still two more steps to go. Once the requirements have been made explicit, we have to *determine* which risks there are. And finally the risks must be *controlled*: measures should be taken.

Summary: in five steps the needs will be defined, valued and worked out in demands and wishes, after which the risks will be mapped and measures taken to manage these risks. These steps have to be taken for each risk area.



Fig. 3. The Five Steps in a Project for both Product and Process

QTM Matrix

The QTM matrix is created when we match the four risk areas (Functional Requirements, Quality Aspects, Planning & Control and Outside factors) and the five steps (Identify, Value, Specify, Determine and Control):

area	steps				
	1 Identify	2 Value	3 Specify	4 Determine	5 Control
Functional Requirements					
Quality Aspects					
Planning & Control					
Outside Factors					



Fig. 4. The QTM Matrix, in rows the four risk areas and in columns the five steps.

For a project carried out using this method, each cell of the matrix contains specified actions. It is possible that a certain cell is not filled up, but in that case there is a well thought out reason: you choose consciously to take no action, you are aware that you take a risk.

Looking at how projects are executed: they are all focused on controlling the process in terms of time and budget. Of course, in these methods Functional Requirements and Quality Aspects are also important, but the main goal is still: finish the project within time and budget. The quality (of the deliverables) plays second fiddle.

In practice the system delivered (on time) often does offer most of the functionality required. The product is (not completely) fit for purpose and does not meet the expected quality. Conclusion: the customer is not satisfied.

QTM will help you to find the right balance between attention for process and product. The QTM matrix addresses the relevant issues regarding customer satisfaction in a structured way to ensure optimal result in each area. Take care you get the demands and wishes well mapped, determine the risks and the consequences of those risks. When you know the consequences, you can decide how much time and money you want to spend on controlling the impact of the risks.

With the overall picture of QTM in mind, we will look in detail to the activities required when defining the product specifications in order to assure that the product will meet the expectations. How to deal with the implications of the product specifications for the process and possible measures required to achieve the goals will be mentioned. The improvements in the project can be seen as start-up for more general process improvement later on.

The first step to take is finding out what the product at least should do.

3.2 Functional Requirements

The Functional Requirements comprise all functionality that the information system should offer in order to support the business process: the IS should at the least be fit for purpose. QTM supports this row in the QTM Matrix with a characteristic approach.

Step 1: Identify Functionality

The needs of the various customers will be identified in group sessions. Representatives of the customers sit together to identify the business requirements. The requirements will differ depending on the role and the interests of the various participants. To control this session the scope, constraints and budget (time and/ or money) of the IS must be set clearly. If these items are not clear, the uncontrollable increase of demands and wishes, is a serious risk. Another thing is, is the functionality technically feasible. IT needs to keep an eye on this aspect of the requirements. When it is necessary to discard a large part of the functionality in the next steps it will be very difficult to keep the user satisfied. It is therefore useful to introduce metrics at this stage. Rules of thumb based on Functional Sizing metrics like function point analysis and correlated productivity figures can help to set clear limits to functionality. At the end the result of these sessions is an enumeration of functionality that reflects the requirements of all customers.

Step 2: Value Functionality

The importance of the functions will be valued from the point of view of relevant customers like:

- the business: how important is the function for the business process?
- the user: which functions are important for operational tasks?
- the administrator: which functions are necessary to manage and maintain the system?

The type of customers participating is not limited to the three customers mentioned. Depending on the (strategic) position of the new IS other disciplines can be involved in these sessions. The discussion and valuation take place in group sessions, as much as possible. All participants have to agree upon the ultimate product. At the end the result of these sessions is a valued enumeration of functionality that reflects the requirements of all customers. And possible a rough idea about the size of the product and budget required producing the product.

Step 3: Specify Functionality

The user(s) specifies the main functional aspects in such a way that the developer will have a good picture of the system that should be built. In group sessions the requirements will be drafted in headlines or written out in detail by the individual customer. Activities depend on the status of the project. The first real involvement of IT will be in this step.

Step 4: Determine Risks

After the specifications are drafted, the (standard) risks are considered: the consistency of the requirements (several translations depending on the development method) and the quality of the data (reliability, actuality and conversion).

Step 5: Control Risks

This will be explained in chapter 4 - Improved Product Quality by Risk Control.

3.3 Quality Aspects

When discussing Quality Aspects all “other” properties of the information system are included. Aspects that can be seen as the Human factors of the system. The right attention to the quality aspects makes the information system more than just fit for purpose. In fact these aspects will have a considerable influence on customer satisfaction.

Getting clear what the system should do is challenging. Even more challenging is to find out what customers really expect of the way the IS should reveal its functionality. QTM tackles this issue in the same way as it does with functionality by passing through all five steps.

Step 1: Identify Quality Aspects

It is very difficult to start identifying quality aspects without any reference. With functionality you can fall back on your business but identifying quality is another story. Fortunately we can use international standards like ISO-9126 [4], Quint II [5] or TMap® [6]. These standards give you a list of attributes (and definitions) that helps you to define and to decide about quality.

In QTM part of this step is the translation of the definitions in propositions that are easier to understand for the relevant customer. Of course you can use the propositions from earlier projects or from other companies. A good understanding of what is intended by a quality aspect is relevant in the next step. In principle any (standard) list of quality aspects can be used as a reference. Using an international standard offers the possibility of comparing the propositions or interchanging propositions. The ability to understand the quality aspects will improve.

Step 2: Value Quality Aspects

When valuing quality aspects, all attributes from the chosen list that are relevant for the required information system will be investigated. We use at least the same points of view used when checking Functional Requirements: Business Importance, User Importance and Administrator Importance. Each customer has a different view of the system and has different (sometimes conflicting) interests. Sometimes even more parties with specific requirements are involved e.g. Accountancy and Security. You

¹ TMap® Test Management approach is a registered trademark of IQUIP Informatica B.V.

have to aware that when the number of customers of the system increases, also each customer expects to get what he or she wants and need to do the job.

It is difficult for each customer to indicate what is important. Based on a list of quality attributes is almost impossible. In short, choices about properties and valuing are not so easy.

QTM uses a tool to value the properties: the Quality Radar. The first activity is to enter the list of quality aspects and the corresponding propositions in the tool. The next action is to store the relevant customers of this project. The last step of the preparation of the tool is linking the relevant quality aspects to the defined customer.

Then it is up to the customer, he or she must value the quality aspects. To support this process of decision making, the tool generates for each participant a questionnaire. Each question in the questionnaire consists of two propositions from which one must be selected. Deciding with the required information system in mind, the customer indicates the relative importance of the “other” properties of the system. The output of the questionnaire will be shown in the Quality Radar. The fig. 5 shows the result of a questionnaire using Quint II as a reference.

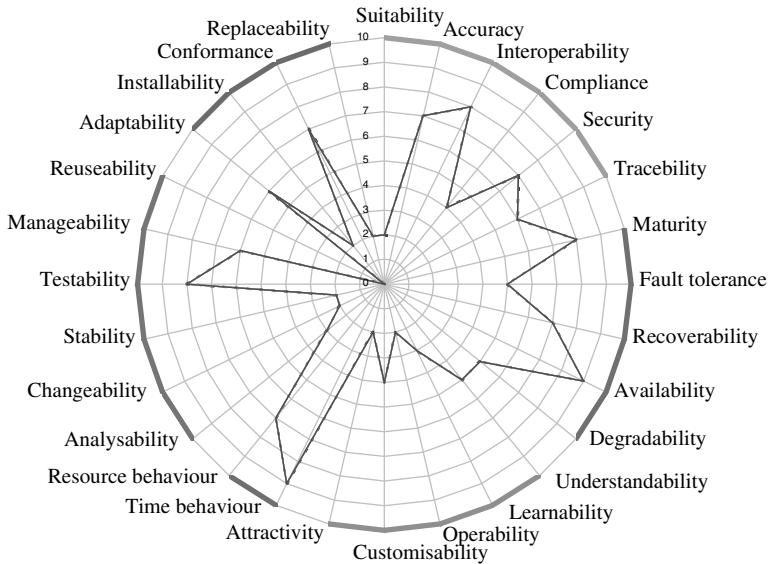


Fig. 5. This plot shows a possible result from a Quality Radar session. In the plot you read that Availability is more important than Stability. The output shows that there is no consensus on various issues, so we need to discuss a lot with the customers.

The radar shows in one plot how each individual, a specific customer (one or more representatives) or all customers think about quality. It gives an indication which quality aspects are important and that some of the quality aspects are more important than others. It shows also differences in perception between participants of a specific customer and differences between customers. The Quality Radar proved to be a powerful communication tool: it gives a good basis to discuss the quality issues of

system and leads to greater involvement of all participants. It helps to create consensus about the “soft requirements” (the human factors) of the product.

Step 3: Specify Quality Aspects

The (most) important quality aspects must be specified. If “Availability” is considered to be important, the customer must define concrete requirements, e.g.: “the information system must be operational five days per week, eight hours a day”. The properties must be defined in such a manner that during the project, it can be checked whether the IS will meet the specified requirements.

By “Operability” we mean the customers want an easy to handle information system. This requires for example uniform screen layouts, not only within the system, but also common with other existing systems.

Step 4: Determine Risks

When the properties are known in detail, the developer has a good understanding of what should be built. Further, he has to think about what kind of risks he runs in developing the system? Is the requirement a certain transaction should be completed within two seconds feasible? Is it easy to accomplish, or is it a problem - due to the way the transaction is defined, the performance of the network, etc.? If it is a problem, measures must be taken to meet the specified requirements.

Step 5: Control Risks

This will be explained in the next chapter.

4 Improved Product Quality by Risk Control

4.1 Product Risk Matrix


Step 5: Control Risks

The project already passed the steps: Identify, Value, Specify and Determine for both functionality and quality aspects. So the attention points or risks are known and now it is time to define the measures.

An essential part of QTM is the direct linking of measures to products. Here we don't mean the ultimate product, the information system. At this point we mean the deliverables of the process to produce the product. You can't make product without a production process. To be able to control that process some intermediate products or deliverables are defined. This list of deliverables depends on development method and the implementation of the method in an organisation, but it is always possible to draw up a list of deliverables. This list does not contain only documents or coding; for instance a “prototype” is also a kind of intermediate product. Mind that products like “Functional System Design” are too generic; the matrix works better when products comprise smaller logical units, such as function structure, data model, user interface

and a description of the administrative organisation. Only the deliverables that describe the product will be included in the matrix (system documentation), not the process documentation (like plans, phase reports, etc.).

In the Product Risk Matrix the columns reflect the possible product risk areas (Functional Requirements and Quality Aspects) and the rows, chronologically represent the deliverables of the project. In the Product Risk Matrix you link the risks directly to deliverables. So it easier to define concrete measures or to position the measures and the results will be visible on the right moment.



Product Risk Matrix

	Functionality				Quality Aspects						
	consistent definitions	quality of data	inefficiency in the project	exceeding of the budget	standard screen lay-outs	context related help screens	at least 50 users in one session	authorisation of transactions > 50k	screen changes < 0.5 seconds	couple to financial admin.	data definitions as in business model
Organization Model								✓			
AO procedures								✓			
System Architecture							✓			✓	
Overview functions	✓										
Overview data	✓	✓								✓	✓
Function / Data Matrix	✓										
Man Machine Interface					✓	✓					

Fig. 6. The Product Risk Matrix. The marks in the cells indicate in which product (rows) a determined risks (columns) need to be checked.

For the attention points the concrete requirements are known. To assure that the product will meet the requirements we have to define some (control) measures. In the Product Risk Matrix we mark (✓) the points in which (intermediate) product we can check that the requirement for the (intermediate) product will be met. By these marks the requirements are linked to the deliverables of the project. The earlier certainty is given in the project the better. With the Product Risk Matrix you can tune the measures, actions take place at the appropriate time: optimising activities and controlling risks.

The Product Risk Matrix is the basis for the specification of product related measures. In the matrix concrete measurable requirements are linked directly to physical project products. The (intermediate) products will be used early in the project to ensure specified properties of the end product. The project leader can take now appropriate measures based on a risk analysis. All measures intend to improve quality of product and process.

5 Project Improvement

5.1 Quality Plan

Ultimately QTM leads to a project improvement, not only process improvement on project level but also product improvement. Based on the Product Risk Matrix the project approach will be adjusted. Briefly, after defining the definitive list of deliverables the strategy must be defined, while the measures must be specified in a quality plan and/or a risk plan:

- Agree upon product standards: requirements will be seen in at least one deliverable.
- Schedule reviews, checks and test activities;
- Decide on the techniques to be used, dependent upon the products to be reviewed, the manner in which those products will be reviewed and which techniques fit best.

Choices should be made based on the Product Risk Matrix that shows the total overview of the risks and the relation to (intermediate) products: at which moment and by who will the checks be executed, on which (intermediate) product, and what will it cost? Decisions can be made, together with all participants on the basis of a cost / benefit analysis: do we take this measure or not? It is the responsibility of the project leader to decide if it is more efficient to combine test activities, to add additional products and so on.

5.2 Process Control

The development process usually starts with drawing a plan. In this plan the goals for the project are set and the milestones in the process. The five steps are also very useful making this plan. The project leader identifies the issues that are important for the project, e.g. development platform, organisation, schedule, resources, security-plan, quality-plan, etc. With the customers the issues are valued and specified.

The most important items in this area for customers are Time and Money: when is it finished and what will it cost? The availability of metrics helps us to estimate effort. Effort can be expressed in time or in money. We can do this on account of the correlation between functional size and effort [6, 7, 8]. Another main variable of effort is the development platform and the available tools. The collective noun for all effort-related variables is productivity attributes.

After that the project leader should perform a risk analysis (determine) and define the appropriate measures to attend to the risks (control). At the end, all of this should be reflected in the project-plan.

6 QTM in Practice

6.1 Introduction

Most improvement programs starts in the top of an organisation. At the management level it is decided that the organisation should go to level x of CMM(i) or the next level of EFQM. It starts with an assessment and based on the outcome the objectives are set, time schedules are defined and so on. For the long term this top-down approach is a correct way to start process improvement in large organisations. For immediate or short time result this is often not the right approach.

Projects are running and the customer wants value for money now. This requires a bottom-up approach. QTM offers you such an approach. Starting point for a project is the development process of the organisation. Based on product specifications and the Product Risk Matrix the development process is tailored for that particular project. During the project and at the end of the project the adjustments (improvements?) will be evaluated. When adjustments of the process gave good result why not implement this adjustment(s) in the regular process. When an overall improvement program is already defined, adjustments should fit in this overall program.

The ideal situation is of course when you introduce an approach or method like QTM at the start of a project. This will avoid some inefficiency during the project. In QTM you can fill in the activities in all the cells of the QTM matrix tuned on the wanted situation. In practise however projects are already running when the need for “real” quality assurance is felt. When the customer realises the result of the project is not meeting the expectations (functionality, quality aspects and process control) action should be taken. The power of QTM is that it is useful at that time as well.

6.2 The Case of a Small Insurance Company

In this case, Software Control (SC) was asked to assist IT-management. Part of the assignment was to improve overall quality of the IT-department. Budget was limited and most of the projects were under time pressure. This meant that the first priority was to help project management doing their job better. We used QTM as a starting point to find out what was the best thing to do in each of the projects.

The first thing we did was finding out the status of each project. “Colouring” the QTM Matrix is a good start. Due to time constraints we limited ourselves to the three major projects. Fortunately all projects were small so it was relatively easy to determine the status. Interviews with IT management, the project leader, a team member and a customer were the basis for filling in the QTM matrix.

The first project was not a project in the common sense. The system was already operational but there were a lot of change requests. Analysis of the requests gave as a result that functionality was not the main issue. Most things had to do with not fulfilling the expectations related to the “human” factors of the system, the Quality

Aspects. In production the customers found out what they really wanted. Lesson One: try to find out earlier what is required. Another thing we discovered is that the customer was pleased with the changes made until now. The change process however, caused the system to change several times a week. A finished request for change was implemented almost immediately. For the customer it was difficult to become familiar with the system. Due to this way of working it was also difficult to control the process. In other words improve the change process in such a way that project control is easier and the customer gets better-controlled implementations. The information gotten from the interviews with management, developers and customers we used to color the cells of the QTM Matrix. The QTM Matrix indicates which areas need to be improved.

	1 Identify	2 Value	3 Specify	4 Determine	5 Control
Functional Requirements					
Quality Aspects					
Planning & Control					
Outside Factors					

Fig. 7. The QTM Matrix colored for each of the projects. For this organization all three projects showed the same result.

The answer to most of the mentioned problems was quite obvious: start with release management. Together with management, developers and customer we defined a procedure for release management. One of the issues was prioritizing the change requests. We started using the steps of QTM to find out the priority of the changes (Functionality and Quality Factors). With Function Point Analysis in Maintenance [8] we created a quantitative basis for effort estimating. Based on required effort and priority the Change Board started with capacity planning to control the size of the release (Planning & Control). Besides covering the regular situation, an emergency procedure was defined for unexpected situations (Outside Factors). The release management procedures that proved to work in this project will be used in future for all other projects.

The second project was in the test phase. Filling in the QTM Matrix gave almost the same result as the previous project. During development there was very little attention to defining test requirements. Due to this, testing was not quite structured. Looking for a (inter) national standard test approach it was easy to find TMap® [5]. TMap® has evolved towards the standard for software testing in The Netherlands. It is being used by more than two hundred Dutch organizations. Most Dutch banks, insurance companies, pension funds and government departments use TMap partly or as a whole. We implemented the most relevant parts of TMap® and used QTM to identify, value and specify the major product objects to test. Based on risk analysis we defined a test strategy. With this in mind the test process was improved. We started also to use a incident administration and procedures to control the test process. The lack of attention for specifying test requirements will be avoided in the next projects.

The third project was just started. The results of the other projects showed that the development process had some “grey” areas. Areas like attention for Quality Aspects and test requirements and a reliable planning had not received sufficient attention. We used the QTM steps to do a quick review on functionality and made some small enhancements to the functional requirement. We introduced the Quality Radar to find out what the customer expected from the system in term of quality aspects. In order to be consistent with other improvements we used the list of quality aspects of TMap®. Based on the discussions in the Quality Radar sessions, some of the functional requirements were changed on details now quality was specified. When the basic issues were covered we filled in a product risk matrix (fig. 6). The project plan was adapted to the new insights and we used fortuitous available expertise on functional sizing metrics [7] to improve planning. Although there is a little delay in schedule and a small increase of budget, both the customer and IT department are confident that a better product will be delivered and that maintenance costs will decrease.

7 Project Improvement as Start-Up

In the case we illustrated how to use QTM in a project, even when the project is already started. Depending on the situation you take the relevant measures at the appropriate time based on the product risk matrix. This matrix helps you the keep up with requirements and to control risks, deliverables and activities. Focus on product quality not only helps achieving product quality but also helps to keep the project within budget and on schedule. It also saves time and money later because the number of failures will decrease.

Adjustments needed to satisfy the customer in a particular project could also contribute to other projects. Improvement does not necessarily have to start from the top: improvement can also be started from the bottom.

Quality Tailor-Made as **Start-up for Process Improvement.**

8 Continuous Improvement

Software Control, a division of IQUIP Informatica, is looking at two major areas in ICT: Testing (300 professionals) and Quality Assurance (100 professionals).

When we are asked to do Quality Assurance in Projects most of the time QA has to support the project manager to do the job. The main focus of a project manager is time and money (strongly related to the process) and there is little interest in quality (product and process related). The lack of attention for the quality of the product and experiences from testing noticing the discontent of customer about the quality factors [4, 5] triggered us to develop an approach to get the right balance between product and process. Including product aspects in Risk Analysis and Risk Management guided us to QTM. The approach is developed further based on practical experience in a great number of project in various business environments e.g. banks, insurance companies, distributors, government and others. We just started to do QA based on QTM in an embedded environment.

All Software Control Professionals in Quality Assurance are familiar with this approach. It is part of their basic training. In most of their jobs they use the approach or elements out of it. Comments from the experts in the field and customer satisfaction surveys will help us to develop further. The latest remarkable renewal was the new release in May 2001 of the Quality Radar. Focus for 2001 – 2002 is Functional Sizing and overall Risk Management.

We hope to get now also some feedback from the academic world. A Technical University in the Netherlands included “Kwaliteit op Maat” (Dutch for QTM) in the learning program for information engineering.

References

1. KoM, Boeters, Andre, Noorman, Bert, "Kwaliteit op Maat", Kluwers Bedrijfswetenschappen, Deventer, ISBN 90-267-2579-5, 1997.
2. Dekkers, Ton, "Maximizing Customer Satisfaction", in ESCOM/SCOPE conference proceedings, London, April 2001.
3. Dekkers, Ton, "Quality Tailor-Made", in Quality Week Europe conference proceedings, Brussels, November 2000.
4. ISO/IEC 9126, Information Technology - Software product quality, characteristics and metrics, International Organization of Standardization, 1997-1998
5. Quint II, Zeist, B. van, Hendriks, P., Paulussen, R., Trienekens, J., "Kwaliteit van softwareproducten", Kluwers Bedrijfswetenschappen, Deventer, ISBN 90-267-2430-6.
6. TMap®, Pol, Martin, Veenendaal, Erik van, "Structured Testing of Information Systems", Kluwer, ISBN 90-267-2910-3, 1998.
7. IFPUG (International Function Point User Group), "Function Point Counting Practices", IFPUG
8. Dekkers, Ton, "Van functiepuntanalyse naar integrale functiepuntanalyse", IQIIP, 1994

LIPE: A Lightweight Process for E-business Startup Companies Based on Extreme Programming

Jörg Zettel¹, Frank Maurer², Jürgen Münch¹, and Les Wong²

¹ Fraunhofer Institute Experimental Software Engineering,
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
{zettel, muench}@iese.fhg.de

² University of Calgary, Department of Computer Science
2500 University Dr NW, Calgary, Alberta T2N 1N4, Canada
{maurer, wongl}@cpsc.ucalgary.ca

Abstract. Lightweight development techniques (e.g., Extreme Programming) promise important benefits for software development with small teams in the face of unstable and vague requirements. Software development organizations are confronted with the problem that a bunch of techniques exist without knowing which ones are suited for their specific situation and how to integrate them into a comprehensive process. Especially for startup companies, guidance is crucial because they usually do not have time and money for creating their development process on a trial-and-error basis. This paper proposes a lightweight software process for a specific application domain (i.e., database- and user-interface-oriented off-the-shelf e-business applications). The process originates from analyzing experience from past e-business projects, interviews conducted with industry, and literature study. Expected benefits of this process are cost effectiveness, sufficiently high quality of the end product, and accelerated functionality-to-market. The process is described according to the dimensions activities, artifacts, roles and tools. In addition, this paper includes a description of a lightweight measurement program that is tailored to the characteristics of the described process. It can be used for controlling the project progress during project execution as well as for evaluating the effects of performing the process in a specific organization or company.

1 Introduction

In the last couple of years, one of the major trends for software development organizations was the move towards e-business systems. These years also saw a large number of startup companies emerging in this area. Unlike older software organizations, these startup companies do not have established development practices. Their processes usually are immature and ad-hoc. Often this is coupled with a less than positive attitude towards software engineering practices and, especially, software process improvement initiatives and software metrics collection. In particular, code metrics (such as lines of code, code complexity etc.) and process improvements standards (such as the Capability Maturity Model) are often viewed as obsolete and irrelevant. Many organizations have developed their own ad-hoc methods of

assessing processes and metrics. Other software engineering techniques, such as requirements engineering and semi-formal specification, are only practiced in an abbreviated fashion.

Seeing this attitude coming from many highly qualified and experienced software developers, we believe that it is a result of the business context of startups. Nevertheless, we also believe that there are long-term benefits from the application of software engineering methods, and it is a good idea to use some of them from the very start. To overcome the negative attitude and create a base for future growth, we propose in this paper a lightweight software process for developing e-business applications called LIPE (for: LIghtweight Process for E-business software development).

The process is based on initial results of several interviews conducted with industry as well as on a literature survey and our own experience in e-business software development. The interviews conducted showed two major trends:

1. Current e-business software development focuses on one hand on upgrading legacy business-to-business systems such as EDI (Electronic Data Interchange) to operate over the Internet.
2. On the other hand, the focus is on implementing new e-business solutions from scratch.

Furthermore, a significant amount of e-business development activities consists of integrating third party applications and platforms.

The first trend is primarily seen in older organizations while the second one is often found in startup companies. LIPE focuses on the second area.

A first empirical validation of LIPE is planned for. By then, LIPE's justification is based on its link to the business context of product-focused startup companies as it is described in Section 2. Section 3 of this paper introduces LIPE and defines its activities, products, and the roles of the team members. Section 4 describes a lightweight measurement program associated with LIPE. The last section discusses our results and future work.

2 Implications from the Business Context of Startups

Time-to-market pressures, the small size of the development team, and the interaction with venture capitalists usually govern the business context of a startup:

- **Time-to-market:** Before any revenues come in, a startup has to solely rely on external capital for covering the costs of development and marketing efforts. The burn rate (amount of money spent per month) allows determining the time when the next round of financing needs to be available. Being able to create revenue and earnings moves this deadline more into the future and, in addition, increases the net present value (which is important in negotiations with venture capitalists). As a result, reducing time-to-market is a primary concern for the startup.
- **Size of development team:** Initially, software development groups of startups are rather small. If the company is a spring-off of a research institute or university, the development team usually includes recent graduates and/or students.

- **Venture capital:** Startup enterprises are usually financed by venture capital and private or corporate angels. Often, the target of venture capital companies is to have an initial public offering of the startup within two to four years. In this timeframe, early stage companies are going through several rounds of financing and substantial increases in size of the development organization. Each round of financing usually increases in size quite dramatically. Additional rounds of financing are not guaranteed from the beginning. Hence, startups see venture capital companies as one of their targets for marketing and they “want to keep the venture capitalists happy”. To encourage the venture capitalists to kick in the next round, the startup is required to show strong progress concerning the software product and/or concerning revenues. As a result, the software development organization focuses at least for the very beginning on producing visible results in the short term instead of a long-term perspective. In the end, there will be no long-term perspective if the next round does not kick in.

The business context described above has several implications on the software development process.

First, due to the time-to-market constraints and the focus on visible results, startups postpone documentation effort into the future. The focus is on producing executable code, not design documentation. The small size of the development team enables this lack of documentation: As all developers work closely together, they replace time spent on formally documenting designs and decisions by time spent on informal communication. As long as the team is small, this approach pays off because it is faster to directly talk to each other instead of writing development knowledge down. In addition, direct communication usually deals with existing issues while for producing documentation the writer has to make assumptions on what information may prove useful for the reader. If these assumptions are wrong or if the software design changes drastically, the documentation effort was wasted. As the development organization grows, the time spent on exchanging knowledge about the software product and on training new people increases sharply.

Second, the pressure to produce additional product features often reduces time spent on quality assurance (like inspections and testing). In the long run, this may lead to quality problems and increased maintenance effort. Nevertheless, this product focus makes sense from a business perspective because of the relatively short timelines for additional rounds of venture capital funding.

The lightweight process proposed in the remainder of this paper is based on results from interviews with software developers in the area of e-business software, relevant literature (especially on Extreme Programming [3, 4, 9, 13]), as well as our own experience. It is bottom-up and lightweight. A top down approach would try to enforce a process with emphasis on documentation and long-term applicability from the very start. We start from existing ad-hoc or “natural” processes and try to add a parsimonious structure to have a basis for future growth and maturing of the software development organization. The process is lightweight because it focuses on the production of high-quality code and not on additional documents. The proposed process is designed for developing database-oriented and user-interface-centric off-the-shelf e-business software using the Java Enterprise framework. User interfaces are either web-based or WAP-based. The process assumes that some team members are inexperienced in the software technologies used as well as in software development in

general (recent graduates or last year students of computer science or information technology programs). We also assume that the development team has access to a senior member of the marketing or consultancy group, who represents the customer's side and is able to make decisions on requirements and feature priorities. The process also takes for granted that requirements change when the marketing efforts progress. Another focus of the process is on customer satisfaction and usability. We use scenario-based requirements specification and prototyping of user interfaces for reaching the last two goals.

To have a basis for future improvements and growth of the development organization, the process also defines a lightweight measurement program. This program focuses on progress tracking as well as on software quality. The former is directly linked to time-to-market issues; the latter is trying to avoid long-term quality problems.

3 LIPE: A Lightweight Software Process

LIPE is based on a small set of key ideas. First, all Extreme Programming (XP) practices [3, 9] except for pair programming are considered to be used: Planning game, on-site customer, small releases, metaphor, simple design, testing, collective ownership, refactoring, continuous integration, and coding standards. Second, goal-oriented and parsimonious software measurement and demand-driven inspections have been added to overcome XP's limitation concerning team size in the long run. They also replace pair programming in its intention to achieve high quality since project managers often are hesitant to use pair programming.

LIPE has been modeled with SPEARMINT™/EPG, a process modeling and process guidance tool developed by Fraunhofer IESE.¹ The description of LIPE is based on a small number of intuitive concepts that are commonly used in software process modeling [5, 12, 10]: Activities, artifacts, roles, and tools; product flow between activities and artifacts; responsibility of roles for activities; and usage of tools in activities. Product flow among activities and artifacts may occur in three variants: Activities may use artifacts (i.e. without modifying them), modify artifacts (i.e. change them during use), or produce artifacts (i.e. create or update them). Product flow clarifies the prerequisites and expected results of each activity. In addition, it implies a restriction on the order, in which activities are actually performed. However, product flow does not determine this order. In an actual project, each activity may be performed as soon as its prerequisites are available and until its expected results are available. It is up to project management to schedule tasks by assigning people to roles and activities, where task assignment is guided by responsibilities in the process description.

Fig. 1, Fig. 2, and Fig. 3 give an overview of LIPE. Fig. 1 and Fig. 2 show the technical development activities together with their products and the product flow among them. They are described in Section 3.1. To complement this, Fig. 3 shows

¹ The development of SPEARMINT™/EPG has been in part financially supported by Stiftung Rheinland-Pfalz für Innovation. More information and a free copy of the tool are available at: http://www.iese.fhg.de/Spear_mint_EPG.

additional organizational and management-oriented activities. They are described in Section 3.2. As can be seen, LIPE consists of a small number of activities (14) and artifacts (18) only. In addition, some important roles and some useful tools have been identified and are described in Section 3.3.

3.1 LIPE's Technical Activities

Fig. 1 and Fig. 2 show LIPE's technical activities. They can be divided into four areas: Activities at the top and to the bottom of Fig. 1 form the interface to the customer. Those in the middle of Fig. 1 are essential development activities, whereas those in Fig. 2 add to the system's quality. Each of the areas is described in turn now.

Top of Fig. 1: In "Collect Scenarios", customer and developers sit together and the customer writes down usage scenarios for the system, which are called user stories by XP. Scenarios play an essential role in the process: They specify required functionality; effort is estimated and measured per scenario; progress is measured in terms of finished scenarios; iterations are planned based on priorities of scenarios; and so forth. In the "Acceptance Test" activity, the customer uses scenarios to judge whether the system fulfills the anticipated needs.

Middle of Fig. 1: In "Realize Scenario", "Refactor System", and "Rework Code", the developers write and test Java code (including code for unit test cases), and write any necessary text/markup files (e.g., HTML, WML, XML). However, the starting points and purposes differ: In "Realize Scenario", developers extend the system to provide additional functionality according to the respective scenario, potentially by integrating existing COTS components. In "Refactor System", developers do not change the system's functionality but its internal design [8]. Refactoring supports specific needs or improves maintainability. In "Rework Code", developers fix a defect described by an open issue report or improve the system's non-functional quality as it has been proposed by a non-functional system test report. Though purposes are different, the basic steps performed are the same in any of the three activities: "Write code for automated unit tests (e.g., based on JUnit [9]). Write code for functionality. Write text/markup files. Compile and integrate code. Run unit tests. Release changes (configuration management is mandatory). If a defect is found either fix it immediately, or report it as an open issue." In general, standardized coding styles [as well as some configuration and issue management guidelines need to be followed. The former ensures readability and understandability of the code. The latter ensures that changes to the code are linked to issue reports in such a way that issue-related metrics can be calculated automatically (see Section 4). Both coding style and guidelines are provided by the company's experience base. Initially, the experience base can be a small set of web pages.

² Available at: <http://www.junit.org>.

³ For example, Sun's "Code Conventions for the Java Programming Language", available at: <http://java.sun.com/docs/codeconv/index.html>.

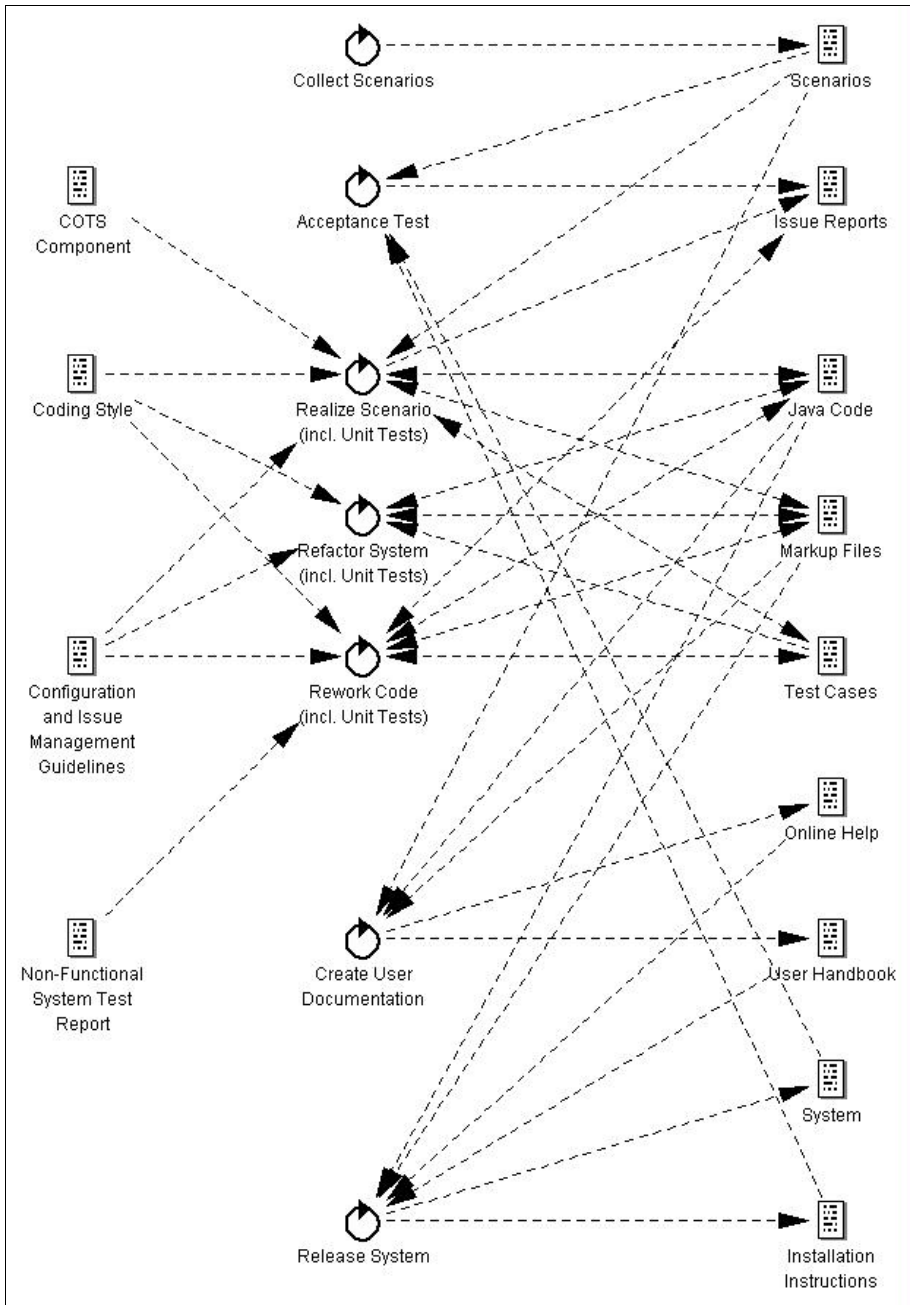




Fig. 1. LIPE's technical activities and product flow among them (Part 1). Notation: Each  represents an activity; each  represents an artifact; each dashed arrow represents a product flow in the given direction.

Bottom of Fig. 1: In “Create User Documentation”, the documentation specialist writes the online help and the user handbook. In “Release System”, the release manager assembles the system (e.g., using an installation wizard tool) and writes the installation instructions. The system – including small incremental releases – is given to the customer for acceptance testing.

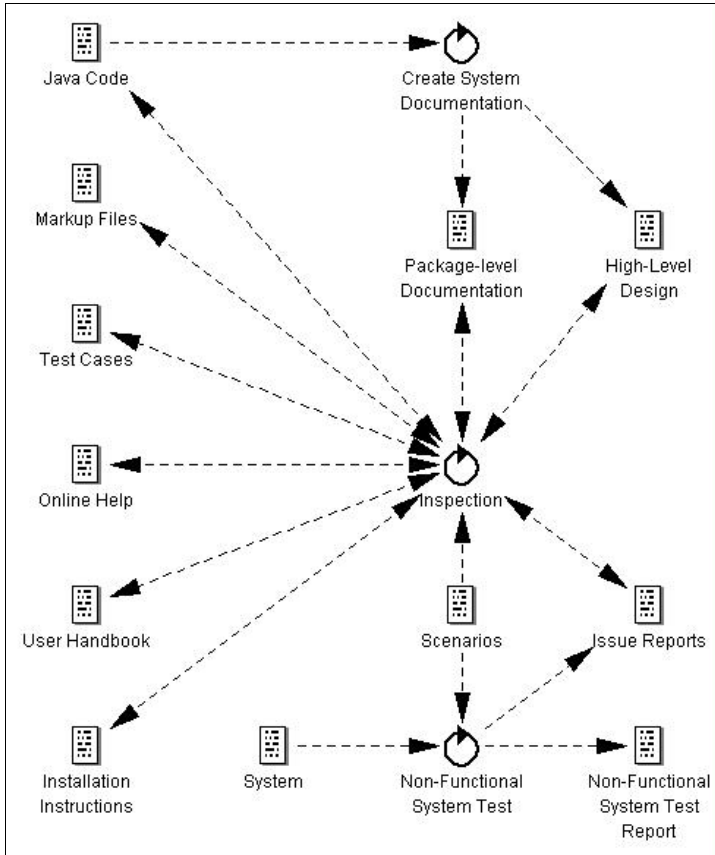


Fig. 2. LIPE’s technical activities and product flow among them (Part 2).

Fig. 2: In “Non-Functional System Test”, the system is tested for acceptable levels of non-functional quality (e.g., usability, scalability, availability [6]) based on what is stated in the scenarios. Insufficient results are documented in a non-functional system test report. System failures are reported as open issues. In “Create System Documentation”, package-level documentation (e.g., using UML diagrams [11]) is written as part of the documentation of the application programming interface (API)⁴ and documents are written to describe the high-level design and/or architecture of the system. In “Inspection”, peers of the author inspect a specific class or text file, and

⁴ It is assumed that the API documentation is periodically generated (e.g., with javadoc).

open issues that have been detected are documented and reported. In contrast to the essential development activities in Fig. 1, the activities in Fig. 2 are performed on demand only. It is the task of the quality manager to decide upon their necessity based on experience, measurement data (see Section 4), and project needs.

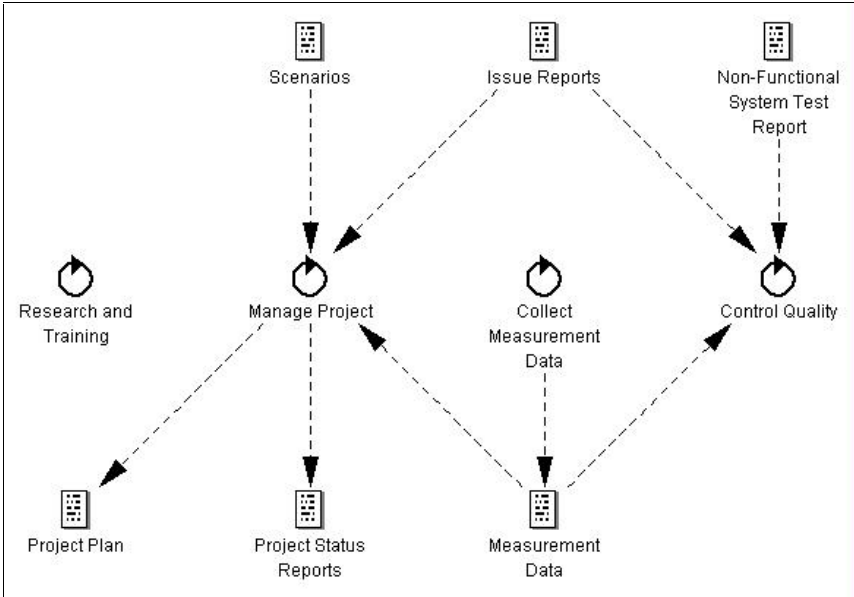


Fig. 3. LIPE’s organizational and management-oriented activities and product flow among them.

3.2 LIPE’s Organizational and Management-Oriented Activities

In addition to the technical activities described in the previous section, LIPE contains a few organizational and management-oriented activities, shown in Fig. 3. In “Manage Project”, the project manager plans, schedules, and monitors the development project based on open scenarios (including effort estimates and priorities), issue reports (which may be defect descriptions, failure reports, or change requests), and measurement data (see Section 4). Planning is actually done by the team as described by XP (see “Iteration Planning” in [9]). In “Control Quality”, the quality manager supervises quality indicators of the system and schedules appropriate quality improvement actions based on experience, measurement data, and project needs. For example: A non-functional system test is necessary if a scenario requires minimum levels of usability, scalability, availability, etc.; writing system documentation may become adequate for very central parts of the system that have reached a sensible level of stability; inspections can be used to look at classes or text files with high issue density or high probability of defects. Especially the latter depends on the availability of measurement data. Therefore, it’s everybody’s task to

“Collect Measurement Data” as part of any other activity. However, measures are collected automatically as far as possible. The last remaining activity, “Research and Training” has been included to reflect the low experience of the anticipated development team and the continuous introduction of new technologies in the area of e-business at least today. Therefore, effort spent on learning should not be neglected but monitored as well.

3.3 LIPE’s Roles and Tools

Table 1 shows important roles and their responsibilities for activities in LIPE. Roles are meant to describe a collection of correlated responsibilities and competencies, not individual people. The project manager assigns each project participant to one or more roles during task assignment. An individual’s skills might constrain such assignments, and assignments to roles change over time, depending on the context as well.

Table 1. LIPE’s roles and their responsibilities for activities.

	Collect Scenarios	Acceptance Test	Realize Scenario	Refactor System	Rework Code	Create User Documentation	Release System	Non-Functional System	Create System	Inspection	Manage Project	Collect Measurement Data	Control Quality	Research and Training
Customer	X	X										X		
Developer	X		X	X	X	X			X			X		X
Documentation Specialist						X						X		X
System Tester								X				X		X
Inspector										X		X		X
Release Manager							X					X		X
Project Manager	X										X	X		X
Quality Assurance Manager												X	X	X

Of all roles listed in Table 1, only the customer role is incompatible with any of the other roles. That is, the individual who is playing the role of the customer may not play any other role in the project. All other roles might be combined, with one exception only: The author of an artifact that is to be inspected may not be one of the inspectors. So, the minimum number of project participants is two people: one customer and one developer/etc.

Table 2 shows some useful tools and their usage in LIPE’s activities. We now give some examples for these tools by looking at open source or freely available tools. The

Java development environment could be IBM's VisualAge⁵ or a combination of Sun's JDK⁶, Emacs⁷ with JDE⁸, and make. The unit test tool could be JUnit⁹. For automatic regression testing of Web-based user interfaces, Empirix e-test suite¹⁰ could be used. Configuration management is integrated in VisualAge, or can be done with CVS¹¹. Issue management can be done with GNATS¹² or Bugzilla¹³. Many metrics tools do exist for different purposes, e.g., JDepend¹⁴ for code metrics. The Apache Software Foundation¹⁵ provides HTTP server, Java Servlet engine, etc. Numerous free databases exist and one compatible with the application server can be selected.

Table 2. Useful tools and their usage in LIPE's activities.

	Collect Scenarios	Acceptance Test	Realize Scenario	Refactor System	Rework Code	Create User Documentation	Release System	Non-Functional System	Create System	Inspection	Manage Project	Control Quality	Collect Measurement Data	Research and Training
Java Development Environment			X	X	X		X		X				X	
Unit Test Tool			X	X	X									
User Interface Test Tool		X	X	X	X									
Configuration Management System			X	X	X	X	X		X				X	
Issue Management System		X	X	X	X			X		X		X	X	
Metrics Tools												X	X	
Application Server		X					X	X						
Database Management System		X					X	X						

⁵ Available at: <http://www-4.ibm.com/software/ad/vajava/>

⁶ Available at: <http://www.javasoft.com/j2ee/>

⁷ Available at: <http://www.gnu.org/software/emacs/emacs.html>

⁸ Available at: <http://jde.sunsite.dk/>

⁹ Available at: <http://www.junit.org>

¹⁰ Available at: <http://www.empirix.com/>

¹¹ Available at: <http://www.cvshome.org/>

¹² Available at: <http://sources.redhat.com/gnats/>

¹³ Available at: <http://www.mozilla.org/bugs>

¹⁴ Available at: <http://www.clarkware.com/software/JDepend.html>

¹⁵ Available at: <http://www.apache.org>

4 Software Metrics

This section describes the derivation of metrics for the following purposes:

1. Measurement data shall be used for controlling progress during project execution.
2. Measurement results can be used to support the application for venture capital by demonstrating the ability to produce functionality in a defined time slot.

Beyond that, the results can be used for weakness analyses of the process to identify improvement potentials. The derivation of metrics is performed in a goal-oriented manner. This enables to concentrate on a small set of relevant metrics and is the basis for performing a lightweight (i.e., parsimonious) measurement program. There are essential differences in measuring key factors (such as effort) in a lightweight process than in a more traditional (i.e., phase-oriented) process. One example for such a difference is the measurement of defect slippage in the context of a lightweight process where inspection activities are demand-driven.

For the description of quantifiable measurement goals we use the Goal/Question/Metric paradigm (GQM). GQM supports the definition of goals and their refinement into metrics as well as the interpretation of the resulting data [1, 2]. The GQM paradigm explicitly states goals so that all data collection and interpretation activities are based on a clearly documented rationale. According to [6], goal-oriented measurement is the definition of a measurement program based on explicit and precisely defined goals that state how measurement will be used. Advantages of goal-oriented measurement are:

- It helps ensure adequacy, consistency, and completeness of the measurement plan and therefore of data collection.
- It helps manage the complexity and reduce the effort of the measurement program.
- It helps stimulate a structured discussion and promote consensus about measurement goals.

In this article, we focus on project control. The goals are to control effort, issues (as defined above), and functionality-to-market. Data concerning functionality-to-market can be used, for example, to get a project trace of the realized functionality, to perform analysis concerning parallel work, or to identify extraordinarily long lasting implementation activities for specific scenarios. In terms of GQM, the goals are:

Goal 1 / 2 / 3: **Analyze** the LIPE-based development project
for the purpose of control
with respect to (1) effort / (2) issues / (3) functionality-to-market
from the point of view of the project manager and the
 quality assurance manager
in the context of the company or organization where the process
 will be applied.

A GQM plan describes precisely why the measures are defined. Besides the goal, it consists of a set of questions and measures. Additionally, models and hypotheses may be part of a GQM plan. In the following we sketch GQM plans for the above goals:

GQM-Plan for Goal 1 (effort):

Quality Focus:

- Q1: What is the effort distribution of LIPE broken down by activities?
Metrics to collect:
- 1) identifier of the activity
 - 2) effort in hours
- Q2: What is the distribution of effort broken down by scenarios and activities?
Metrics to collect:
- 1) identifier of the scenario (or 'overhead')
 - 2) activity identifier
 - 3) effort in hours

Variation Factors / Explanatory Variables:

- E1: What is the experience of the developers with the technology used?
Metrics to collect:
- 1) name of the developer
 - 2) experience level
- E2: How much effort did the developers estimate for each scenario?
Metrics to collect:
- 1) identifier of the scenario
 - 2) estimated effort in hours

Dependencies:

- D1: What influence has the experience of the developers with the used technology on the effort distribution (broken down by activities)?
Hypothesis H2: The effort per activity decreases with the experience of the involved developers.
Metrics to collect:
- 1) activity identifier
 - 2) experience level of developer
 - 3) effort in hours

The motivation for question E2 is to get a better foundation for effort estimations.

GQM-Plan for Goal 2 (issues):

Quality Focus:

- Q1: How many issues were detected in each activity of LIPE (distinguished by source products)?
Metrics to collect:
- 1) issue-id
 - 2) identifier of the detection activity
 - 3) identifier of the source product
- Q2: How many issues were detected in each product of LIPE (distinguished by detection activity)?
Metrics to collect:
- 1) issue-id
 - 2) identifier of the detection activity
 - 3) identifier of the source product

Q3: For each product: What is the distribution of detected issues broken down by issue class?

Metrics to collect:

- 1) issue-id
- 2) identifier of the source product
- 3) issue class

Q4: For each product: What is the average effort and the average calendar time for issue resolution broken down by issue class?

Metrics to collect:

- 1) issue-id
- 2) identifier of the source product (and of the part of the product)
- 3) resolution effort in hours
- 4) start time [dd.mm.yy:hh.mm]
- 5) end time [dd.mm.yy:hh.mm]
- 6) issue class

Q5: How many test cases have been created?

Metrics to collect:

- 1) number of test cases

Variation Factors / Explanatory Variables:

E1: What is the experience of the developers with the technology used?

Metrics to collect:

- 1) name of the developer
- 2) experience level

E2: How new is the basis technology?

Metrics to collect:

- 1) identifier of technique
- 2) maturity level

Dependencies:

D1: What influence does the experience of the developers with the used technology have on the issues?

Hypothesis H1: The number of detected issues decreases with the experience of the developers.

Metrics to collect:

- 1) product identifier
- 2) experience level of developer
- 3) number of issues

The effort for resolving of an issue (Q4) is the effort for the resolution of the issue in the source product and in subsequent products (if affected). Issues can also show up in external products (i.e., COTS components). The resolution effort does not include effort for additional inspections. The issue classification may vary in dependence of the source product. The issue classification depends on the product and the type of issue. An example for an issue classification for code defects is: [{omission, commission}, {initialization, control, interface, data, computation, cosmetic}]. The quantity of created test cases (Q5) indicates a certain level of quality of regression testing and might be used for estimating completion time for a scenario. This could be used as an entry criterion for inspections.

GQM-Plan for Goal 3 (functionality-to-market):Quality Focus:

Q1: For all scenarios: When was the scenario implemented?

Metrics to collect:

- 1) identifier of the scenario
- 2) start time [dd.mm.yy:hh.mm],
- 3) end time [dd.mm.yy:hh.mm]

Q2: For all scenarios: Which incorrect estimations concerning the implementation of scenarios occurred?

Metrics to collect:

- 1) identifier of the scenario
- 2) planned end time [dd.mm.yy:hh.mm]
- 3) actual end time [dd.mm.yy:hh.mm]
- 4) description of the reason

Q3: For all activities: Which dependencies to other activities exist?

Metrics to collect:

- 1) activity identifier
- 2) list of dependent activity identifiers

Variation Factors / Explanatory Variables:

E1: How much effort did the developers estimate for each scenario?

Metrics to collect:

- 1) identifier of the scenario
- 2) estimated effort in hours

Dependencies:

For this GQM plan, we do not describe dependencies because this requires a deeper understanding of the causes for functionality-to-market delays.

Based on these GQM plans, data collection procedures have to be defined. The overhead caused by measurement should be minimized. Therefore, several measures may have to be collected concurrently through integrated data collection procedures. A description on how to define such data collection procedures can be found in [6]. For the above GQM plans, this requires the instantiation of generic attribute types (e.g., product-specific issue classifications) and decisions concerning the point in time, the responsible person, and the best means for data collection. As an example, developers could measure issues always when they document them in an “Issue Report”. A means for collecting data could be an adequate documentation structure for issue entries in the “Issue Report”. A possible template for issue entries in this document could be:

Issue-id:	_____
Discoverer:	_____
Date:	__:__:__
Identifier of the detection process:	_____
Identifier of the source product:	_____
Version no. of issue-prone product:	____
Issue description:	_____
Issue class:	Java Code: () class A.1, () class A.2, ...
Markup Files:	() class B.1, () class B.2, ...
Resolution effort:	____ hours

Finally, measurement data has to be analyzed and interpreted in the context of the measurement goal. For example, a defect baseline could be used to identify products with particularly high defect rates. Possible interpretations might be that the structure of the product is inadequate and needs refactoring, that the developers are not familiar with the enactment of the activity, or that the inspection activity is inefficient. Consequences could be a change of the product structure, training, or a change of the inspection technique (such as providing modified checklists). The quantitative models gained (such as effort baseline, defect baseline, defect slippage model) can be used as a basis for better planning in similar future projects. This creates an organizational learning cycle and, in the long run, a learning software organization.

5 Discussion and Future Work

In this paper, we analyzed the business context of e-Business startup companies and showed how this could be related to a less than positive attitude on standard software engineering methods and procedures. We illustrated why their business context more or less implies an ad-hoc, code-oriented development process. Realizing that these processes do not scale very well, we proposed LIPE as a lightweight development approach that integrates Extreme Programming with ideas from the areas of software measurement for project control and process improvement. We see LIPE as a compromise between ad-hoc, “natural” development processes and more rigorous approaches found in larger software organizations. Following the LIPE approach should provide scalability of the development process over and above the small team sizes for which Extreme Programming was developed and proved successful.

In the future, we are planning to evaluate LIPE in a case study. The startup company that will likely be used for evaluating the benefits of the process has some specific requirements concerning portability of the code, usability, scalability, and availability of their Internet-based software product. These were taken into account when we designed LIPE resulting in activities concerning testing of non-functional requirements. If these requirements do not hold in another contexts, the “Non-Functional System Test” activity can be omitted or the effort spent on it can be reduced.

References

1. V. Basili and D. Weiss. *A Methodology for Collecting Valid Software Engineering Data*. IEEE Transactions on Software Engineering, 10(6), pp. 728-738, November 1984.
2. V. Basili and D. Rombach. "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering*, 14(6), pp. 758-773, June, 1988.
3. Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
4. Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley, 2000.
5. Ulrike Becker-Kornstaedt et al. "Support for the process engineer: The Spearmint approach to software process definition and process guidance." *Advanced Information Systems Engineering: 11th International Conference, CAiSE'99, Proceedings*, LNCS 1626, pp. 119-133. Springer, 1999.
6. L. C. Briand, C. M. Differding, and H. D. Rombach. "Practical Guidelines for Measurement-Based Process Improvement." *Software Process: Improvement and Practice*, 2(4), pp.253-280, 1996.
7. Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. Internal Thomson Computer Press, London et al, second edition, 1997.
8. Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
9. Ronald E. Jeffries, Ann Anderson, and Chet Hendrickson. *Extreme Programming Installed*. Addison-Wesley, 2001.
10. Frank Maurer et al. "Merging Project Planning and Web-Enabled Dynamic Workflow Technologies." *IEEE Internet Computing*, 4(3), pp. 65-74, May/June 2000.
11. Object Management Group (OMG). *OMG Unified Modeling Language Specification, Version 1.3, First Edition*. OMG document ad/ 00-03-01, March 2000.
12. M. Verlage et al. "A synthesis of two process support approaches." *Proceedings of the Eighth Software Engineering and Knowledge Engineering Conference (SEKE'96)*, Knowledge Systems Institute, Skokie (IL), USA, pp. 59-86, June 1996.
13. J. Donovan Wells. *Extreme Programming: A gentle introduction*. Available: <http://www.extremeprogramming.org/index.html>. 6. April 2001.

Evaluation of the E3 Process Modelling Language and Tool for the Purpose of Model Creation

M. Letizia Jaccheri and Tor Stålhane

Norwegian University of Science and Technology,
IDI/NTNU 7491 Trondheim, Norway
{letizia, Tor.Stalhane}@idi.ntnu.no

Abstract. In this paper, we report from an experiment which compared the E3 PML with respect to the standard modelling language IDEF0 for the purpose of model construction. The experiment has been run as part of a software process improvement course in which forty students participated.

Our hypothesis was that E3 will lead to less problems than IDEF0 when constructing software process models. Here, we show how our experiment has validated the hypothesis.

1 Introduction

The research community has developed several software process modelling languages (PMLs) and tools in the last twenty years. Moreover, some attempts have been made to test these languages and tools, also in an industrial context [2] [3].

In the last years we have developed the E3 system which offers a PML and a tool [7]. We have also done some attempts to test E3 both in industrial and in academics settings. This paper describes an experiment¹ we have performed in order to test if the E3 system is easy to use for creating software process models.

Since E3 is designed as a special purpose tool for modelling software development processes, we expected that users would encounter less modelling problems when using the E3 PML than when using a general purpose modelling language.

In order to evaluate E3 PML, we decide to compare it to a standard modelling language and its associated tool support. As a standard modelling language we choose IDEF0 which is based on SADT and is well known in academia and widely used industry.

We designed a modelling exercise in the context of a software process improvement course [6] where 40 students, divided into 10 groups modelled a software process by using one of the two PMLs. Five groups modelled with E3 and five groups with IDEF0. As a quality measure, we choose the number of problems

¹ This work was partially supported by the SPIQ project. SPIQ is a Norwegian project on Software Process Improvement (1997-1999). SPIQ means Software Process Improvement for better Quality.

encountered using the PMLs. We provide both a statistical analysis of problems and a discussion of these problems.

The rest of the paper is structured as follows. Section 2 introduces E3 and IDEF0 (with the associated supporting tool KBSI AIØ Win). It also presents previous evaluation of the E3 system. Section 3 presents our experiment. Our conclusions are given in section 4.

2 Background

2.1 The E3 System

The E3 PML. E3 [7] PML is a formal object-oriented modeling language developed at Politecnico di Torino, specially aiming at process elicitation and process analysis. The representation is graphical. The E3 p-draw drawing tool supports creation and management of E3 process models. E3 PML offers three modeling levels: meta-classes (which is used by system administrators), class or template (to denote general process models), and instance (to express instantiated models or plans). E3 provides five kernel classes:

- **Task:** a class whose instances represent the activities carried out in the process.
- **Resource:** a class whose instances model actual resources with their responsibilities and skills.
- **Data:** a class whose instances model process artefacts, such as source code fragments and invoices.
- **Tool:** a class whose instances represent a specific version of an automated tool or of a written procedure.
- **E3Object:** it is the superclass of the latter four.

E3 also provides kernel associations:

- **Association(C1,C2):** expresses a generic relationship between an object of class C1 and a set of objects of class C2.
- **Aggregation(C1,C2):** expresses the "is-composed-of" relationship between an object of class C1 and a set of objects of class C2.
- **Subtask(T1,T2)** is used to represent the decomposition of an activity into its subtasks. T1 and T2 must be subclasses of class **Task**.
- **Preorder(T1,T2):** denotes the precedence relationship among tasks. T1 and T2 must be subclasses of class **Task**.
- **Responsible(R,T)** denotes the "is-responsible-for" relationship between a resource and a task. T must be a subclass of class **Task**; R must be a subclass of class **Role**.
- **Input(T,D):** identifies the inputs for a task T. T must be a subclass of class **Task**; D must be a subclass of class **Data**.
- **Output(T, D):** identifies the outputs of a task T. T must be a subclass of class **Task**; D must be a subclass of class **Data**. The transitive closure of the **Input** and **Output** associations defines the data flow in the process model.
- **Use(T,TL):** identifies the tools used within a task. T must be a subclass of class **Task**; TL must be a subclass of class **Tool**.

E3 p-draw. The E3 p-draw is a drawing tool that supports creation and management of E3 process models, and provides a mechanisms that enables model inspection according to selected views. E3 p-draw is a beta release and not a commercial product. The tool offers drawing views to develop process models. Figure 1 shows an example of such a view which contains the definition of one of the models developed during the experiment.

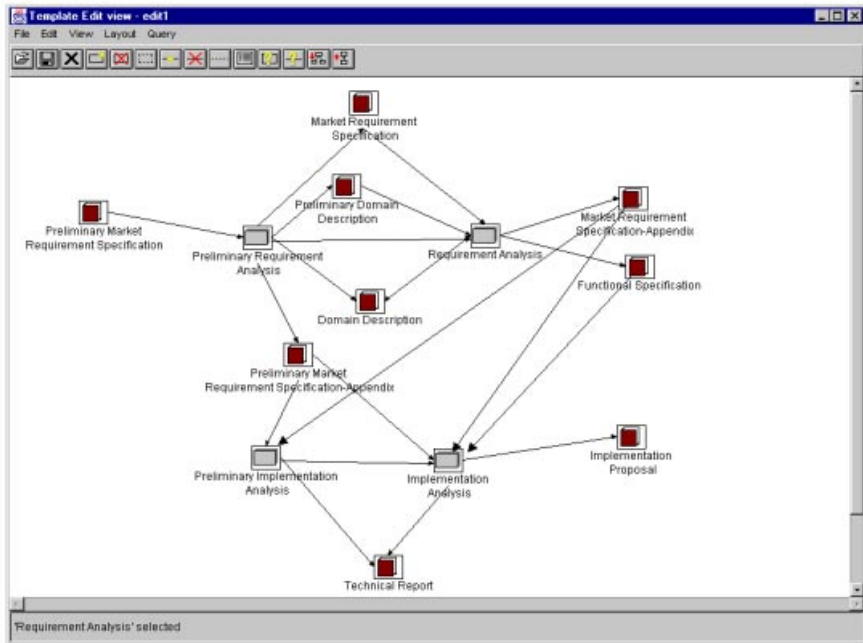


Fig. 1. An edit view displaying one of the five E3 model developed during the experiment.

E3 p-draw also offers several types of derived views or filters:

Simple. This filter is applied to a class (object) to show all associations (links) in which the class (object) is involved, with the exception of the Aggregation and Subtask associations (links). This filter makes it possible to illustrate the entities that are in some way related to a specific entity at the same level of abstraction. See figure 2 for an example.

Composite. When applied to a class C (object O), this filter shows all the classes (objects) that are components of C (O) through the Aggregation and Subtask associations (links). Moreover, it applies the simple filter to each of these classes (objects).

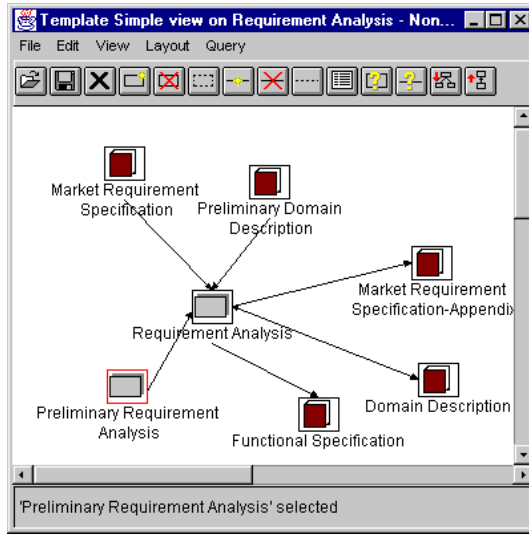


Fig. 2. The simple view to class Requirement Analysis of Figure 1

Recursive composite. When applied to a class (object), this filter performs the recursive application of a composite filter to show all the classes (objects) that are directly or indirectly associated (linked) to the selected class (object) through Aggregation or Subtask associations (links).

2.2 IDEF0 and KBSI AI Win

IDEF0. The IDEF0 function modeling method is designed to model the decisions, actions, and activities of an organization or system. IDEF0 was derived from the Structured Analysis and Design Technique (SADT). IDEF0 does not distinguish between class and instance level. As an analysis tool, IDEF0 assists the modeler in identifying the functions performed and what is needed to perform them. Only input output associations can be defined. The IDEF0-models are based on a simple syntax. Each activity is described by a verb-phrase label placed in a box. Inputs are shown as arrows entering the left side of the activity box while the outputs are shown as exiting arrows on the right side of the box. Controls (denoting control flow) are displayed as arrows entering the top of the box and mechanisms (denoting resource allocation) are displayed as arrows entering from the bottom of the box. The processes can be decomposed by expanding activities.

KBSI AI Win. Knowledge Based Systems, Inc. (KBSI) [5] has developed several tools for the Integration Definition for Function Modeling (IDEF0) family. AIØ Win is one of the tools that uses IDEF0 as the modeling language. Fig-

ure 3 shows a snapshot of an AIØ Win view containing one the IDEF0 models developed during our experiment.

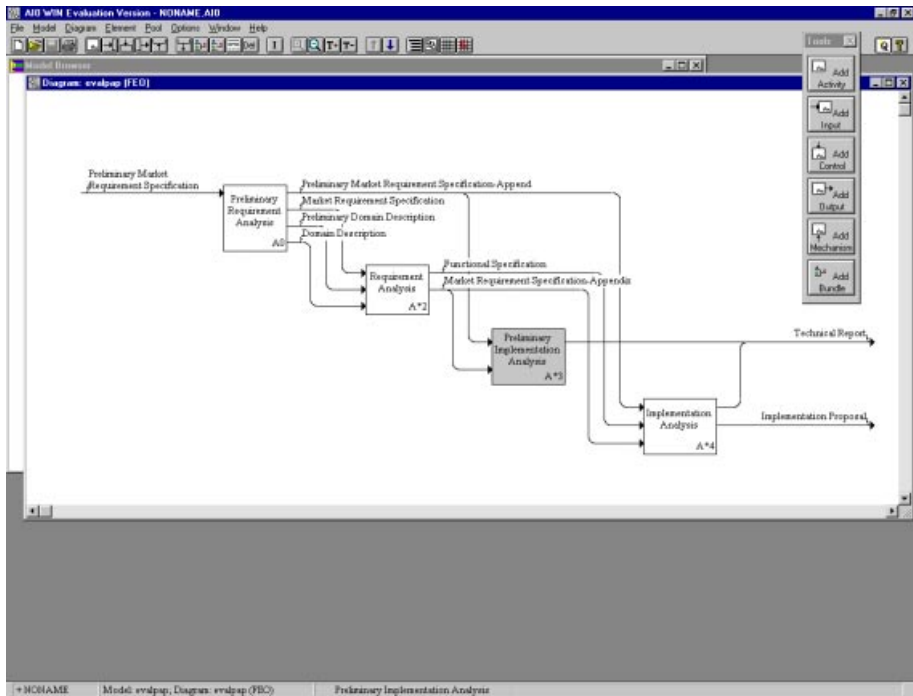


Fig. 3. An AI Win view displaying one of the five IDEF0 model developed during the experiment.

2.3 Earlier Attempts of Testing E3

Testing the ideas. The design of E3 focused on software process description (understanding and static analysis) and the use of object oriented modeling. In 1993, we started by using object-oriented techniques (Coad-Yourdon) to model the software process described in a quality manual we had obtained from Iveco - a division of Fiat. From this first modeling phase and interaction with process owners, we deduced the requirements for the E3 system and we had the chance to test our ideas.

For several years, we have been using the Iveco process as a reference for internal benchmarking of subsequent implementations of the E3 system.

Testing E3 for the purpose of understanding. The E3 system has been used for four years in a software engineering course being taught at Politecnico di Torino to describe the software process used for all student project

activities. The goal of this effort was to test the suitability of E3 description for process understanding.

Testing E3 for modelling of general processes. One process modeler has worked in collaboration with the process owners at a Department of Olivetti [1]. The modelling phase triggered changes in the original process descriptions. The process owners seriously considered adopting E3, but a subsequent reorganisation of the Quality Department terminated this attempt. The modeller belongs to the E3 research team, was well acquainted with the system and managed to overcome the system limitations. Thus, almost no problems were reported.

3 Our Experiment

This section reports our experiment. For the definition, planning, and analysis of the experiment, we have mainly used [4].

3.1 Problem Statement

This work has been performed in the context of a case study in which 40 students (organized into 10 groups) from a software quality and process improvement course interacted with a Norwegian telecom company. During the case study the students came in contact with three actors: the quality manager, the manager of the process group, and a project leader. Among other tasks, the students were asked to model a process fragment, and to report about the problems they encountered. Five groups used E3 and five used IDEF0.

All students were majoring in computer science, with some exceptions; one PhD student and three business administration students. The groups were hence quite homogenous in age and knowledge. The input which was the basis for the case study is shown in Figure 4. The educational objective was mainly an exercise in process modelling and gaining experience with a formal process modelling language.

3.2 Experiment Planning

Since E3 is designed as a special purpose tool for modelling software development processes, we expected that users would encounter less modelling problems when using the E3 PML than when using a general purpose modelling language. Our hypothesis is thus that:

H1: For the purpose of creating software process models, the E3 PML is easier to use than a standard modelling language and tool.

This can be rephrased as:

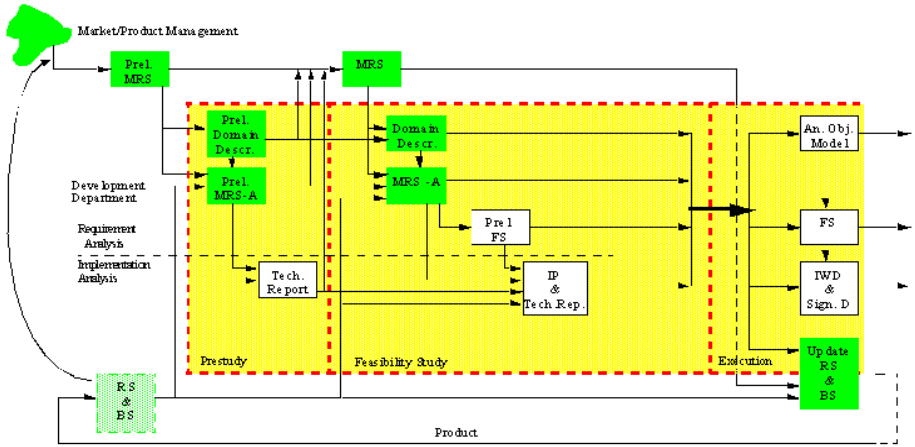


Fig. 4. A view of the input process description.

For the purpose of creating software process models, the average number of modelling problems (μ_1) that students encounter when using E3 (PML and tool) is less than the average number of modelling problems they encounter when using IDEF0 (μ_2). We state the null hypothesis as:

$$H_0: \mu_1 = \mu_2$$

We have one factor (PML and tool) with two treatments (E3 and IDEF0). The student groups are the subject of the experiment and the dependent variable is the number of modeling problems encountered.

3.3 Experiment Operation

As mentioned before, we have decided to use students as subjects of our experiment. The reason for our choice is that it is difficult to convince industry persons to use the tools we propose. Moreover, it is almost impossible to convince them to perform the same task several times.

We motivated students by providing them with a real process (see figure 4) instead of just a toy example. The process was presented by a person from industry who also was aware of the experiment and communicated to the students the importance of performing such experiments. Concerning experiment execution and data collection, we had at least two choices:

- Observing the students while developing the process models. This could be supplied by video registration.
- Ask the students to list the problems encountered as part of the report they had to deliver after the exercise.

We chose to ask the students to write a list of the encountered problems. The reason for rejecting a direct observation of the student work, was that it would have been time consuming. Also, the students did not always solve their exercises at the university and not always during normal working hours.

Most of the groups had some problems interpreting the process model for the company. Four out of ten groups reported that they had problems interpreting it. These groups are not the same as the ones that reported problems interpreting the model. This indicates that the input description was not good enough. The underlying information from company was vague. More interaction after a first modelling session may have helped. An alternative would have been to enable more interaction between the students and the company by letting the students present the results directly to the company.

3.4 Presentation of the Data

The entire set of collected data is presented in table 1.

If we look at the reported problems in 1 we notice that most of the IDEF0 problems are related to *complexity* while E3 problems can mainly be related to *usability*. It seems that it is more complicate to map a software process to a model using a general purpose modelling language than using a process specific language and tool, like E3. Table 2 displays the raw data on which we have performed statistical analysis.

3.5 Experiment Analysis

We choose a non-parametric analysis in order not to make strong assumptions about the data. We started by ranking our data so that the biggest number of problems get the lowest ranke. The result is shown in table 3.

Table 3 is organized so that the less problems a user has with a PML, the higher ranke sum it will get. We observe that the sum of ranks (called W_s) for E3 is 33 while the sum of ranke numbers for IDEF0 is 22. From this, it appears that E3 PML is better than IDEF0.

The data analysis is performed by Wilcoxen analysis.

$$E(W_s) = \frac{n}{2}(n+1)$$

$$Var(W_s) = \frac{nm}{12}(N+1) - \frac{mn \sum_{i=1}^n d_i(d_i^2 - 1)}{12N(N-1)}$$

Here n is the number of elements in the group which is the base for W_s and m is the number of elements in the other group. $N = m + n$. e is the number of groups with identical values while d_i is the number of identical values in group i.

For the data set displayed in table 2, we have $n = m$. This gives $E(W_s) = 27.5$.

Table 1. Collected data

Group	Problems with PML
1 (IDEF0)	None
2 (E3)	None
3 (IDEF0)	1. What was complicate with the modelling activity was to decide whether a given influence on a process should be interpreted as control or input to the process. 2. We had problems to specify resources to activities and subactivities precisely. 3. We have used the general concept resources as we did not have available more precise concepts. 4. We found that the constraint that one must have between 3 and 3 subactivities in a IDEF0 model limiting in a case in which we wanted to have two sub-activities. 5. Sometimes there can be very many arrows between the different boxes even if we only have 6 boxes. Such big quantity of arrows makes the models more difficult to follow and to manage.
4 (E3)	None
5 (IDEF0)	1. Some activities has many inputs and this makes the model over-complex.
6 (E3)	None
7 (IDEF0)	None
8 (E3)	1. The problem is the overview. Although with a rather simple process like this one, it is difficult to maintain control. 2. The fact that one must model both orizontal and vertical relationships in addition to document flow contributes to this.
9 (IDEF0)	1.It is difficult to decompose activities. 2. We had problems to distinguish between constrains and input. 3. the model soon becomes over-complex, especially when one has many inputs and outputs
10 (E3)	None

Table 2. Raw Data: Number of logical problems encountered by the 10 groups.

PML	Number of Problems				
IDEF0	0	5	1	0	3
E3	0	0	0	2	0

Table 3. Ranked data.

IDEF0	n. of problems	0	5	1	0	3
	ranke	7.5	1	4	7.5	2
E3	n. of problems	0	0	0	2	0
	ranke	7.5	7.5	7.5	3	7.5

The standard deviation (σ_{ws}) is then 4.25. Moreover, we will use the following approximation:

$$\frac{W_s - E(W_s)}{\sigma_{ws}} \sim N(0, 1)$$

We can use this approximation to test if the observed ranke sum W_s is so much higher than its expected value that we can say that it is not a coincidence.

When inserting the computed values of W_s , $E(W_s)$, and σ_{ws} , we find a test value of 1.29 giving a 10% significance. In other words, there is a 10% probability that $W_s > 33$ even if there is not any difference between IDEF0 and E3.

4 Discussion and Conclusion

As a conclusion from our data, we are 90% sure that there will be less modelling problems when using E3 PML than when using IDEF0 for the purpose of creating software process models.

Concerning statistical analysis, there are three risks stemming from the proposed conclusion:

- We have counted problems with the assumption that all the problems are equal. It could be the case that the two problems observed with E3 are extremely serious while the nine problems observed with IDEF0 all are small. After a study of the reports delivered by the students, we decided that problem counting is acceptable.
- We use a normal distribution approximation in the statistical test although we have few data (only five observations).
- Six of ten observations are identical. This also makes the normal distribution approximation unsure. There will, however, always be a big number of persons who will not have problems when using any of the languages. In this way it is not possible to get an experiment result without a big number of 0-observations.

We thus have to treat our conclusion "there will be less problems when using E3 PML than when using IDEF0 for the purpose of creating software process models" with some reservation. We need more data to increase our confidence in the conclusion. In addition, we should extend the data collection by letting the subjects:

- Register the time they used to model the process. If the time is fixed we should evaluate how much of their task they finish.
- Register the seriousness of each problem, for example on a three value scale (low, medium, high).

We have learned two lessons: first process modelling tool evaluation. Is it at all useful to compare two tools? This kind of comparison is useful for an

organization which wants to choose between two tools. But does it make sense to compare them in an experiment setting?

Second, we have used course students for research evaluation. Is it ethically correct? I.e., can it affect education in a negative way? How can we be sure that we do not influence students for example by giving them better instructions in the tool that we want to prove to be better? Last, how can we make sure that students do not cheat when reporting?

References

1. Fuggetta, A., Jaccheri, M.L.: Dynamic partitioning of complex process models. *Journal of Information and Software Technology* 42 (2000) 281-291.
2. Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, Picco, G.P.: Modeling and Improving an Industrial Software Process. *IEEE Trans. on Software Engineering* Vol. 21. Num. 5. (1995) 440-454.
3. Barghouti, N.S., Rosenblum, D.S., Belanger, D.G., Two Case Studies in Modeling Real, Corporate Processes. *Software Process: Improvement and Practice* Vol. 1. Num. 1. (1995).
4. C. Wohlin et. Al., *Experimentation in Software Engineering An Introduction*. Kluwer Academic Publishers Boston/Dordrecht/ London (1999).
5. Knowledge Based System Inc., Knowledge Based System Inc Web page. <http://www.kbsi.com/> (1999).
6. Dyngsøyr, T., Jaccheri, M.L., Wang, A.I., Teaching Software Software Process Improvement Through a Case Study. *ICECE'99, International Conference on Engineering and Computer Education* (1999) August.
7. Jaccheri, M.L., Picco, G.P., Lago, P., Eliciting Software Process Models with the E3 Language. *ACM Transactions on Software Engineering and Methodology* Vol. 7. Num. 4. (1998) 368-410.

Describing Fractal Processes with UML

Harald Störrle

Ludwig-Maximilians-Universität München
stoerrle@informatik.uni-muenchen.de

Abstract. Component-based software has a self-similar structure on each level of abstraction, i.e. its structure is fractal. Traditional software processes, however, have a linear or iterated structure, and are thus not very well suited for component-based software development. Here, processes described by languages of patterns fit better.

To ensure a general understanding and easy applicability of processes patterns, I propose to (1) build on the well known description schemes for traditional product patterns and adapt them to the software process domain; and (2) use the description techniques and notions of the Unified Modeling Language (UML) is “*the lingua franca of the software engineering community*”. Some adaptations and extensions become necessary to both of these, however, and care has to be taken not to impede the goal of universal understandability.

1 Introduction

1.1 Motivation

Workflows concerning software processes have been described in a variety of formalisms and styles (see e.g. [17] for a survey). Today, some essential requirements are still not adequately addressed by many of them.

Understandability Practitioners explicitly ask for “*well-structured*” process models that are “*easy to understand and easy to apply*” (cf. requirements 2 and 3 in [19]). The vast majority of traditional processes like the “Unified Process” (UP, cf. [15]), however, are rather big sets of documents that take a lot of experience to understand and apply properly.

Flexibility Many developers feel that their creativity is stifled by too rigid a development process. In fact, some would argue, that this is exactly the purpose. Anyway, enforcing a process is sometimes a difficult managerial task, and there are circumstances where traditional (large scale) processes just aren’t appropriate, as is shown by the recently soaring interest in so called lightweight processes like “extreme programming”.

Precision On the other hand, a development process must be formal enough to allow for automatic enactment and formal analysis, at least in selected places. Ideally, this would be combined with the previous requirement to achieve a kind of controlled and graded formality of the process.

Fractal structure Finally, the componentware paradigm has manifest implications on the overall structure of the development process: if the product structure is fractal, the process structure better be fractal as well. Otherwise the mapping between process and product alone becomes a major challenge. Iterative process models, such as the UP have a crippling weakness here.

Integrate aspects Classical process models (e.g. V-Model, ISO 12207, UP) differentiate between different aspects (aka. submodels, core workflows). These, however, really are tightly interwoven. Separating them makes it easier to neglect the supportive workflows like risk management or quality assurance, as these do not contribute directly to the (amount of) output.

1.2 Benefits of approach

I believe, that process patterns using UML are a novel way for dealing with these issues that is superior to traditional approaches for the following reasons. First, consider the contribution of the UML. On the one hand, the UML is already “*the lingua franca of the software engineering community*” (cf. [25, p. v]), and it is very likely to spread even further. This means that almost every professional understands UML (or will have to do so pretty soon), and things will stay like this for some time to come. This results in network effects, rapidly increasing the number and quality of CASE-tools for, courses on, and experience with UML. By using the UML, the understandability of process models is increased dramatically. Some object that the UML is still not very expressive when it comes to modeling workflows, but with UML 1.4, this is can be solved without heavy weight-extensions.

On the other hand, UML is now rapidly being developed into a body of mathematically precise formalisms (cf. [23, 18, 26]), opening up the road to enactment and formal analysis, not only of process models, but also of the product models.

Now consider the contributions of process patterns. The notion of (design) pattern is widely accepted among practitioners, and has resoundingly proved its practical applicability. Patterns capture small, coherent, and self-sufficient pieces of knowledge, thus allowing to be applied in different contexts, and in particular, on different scales. Using (design) patterns is a form of reuse of (design) knowledge.

Process patterns are exactly like design patterns, except that they exist in the process domain. Actually, the term “design pattern” is a bit misleading: what classical design patterns like Model-View-Controller etc. really talk about is (only) the *product*, that is, the *result* of the design, not the *process* of designing it. So, I shall speak of result patterns vs. process patterns to make clear the difference.

The benefits of design patterns may be transferred to the process domain: by using the same pattern in different contexts, pattern-oriented process-descriptions are less redundant than traditional ones, and so a pattern language is easier to understand and apply than traditional processes. Also, each pattern may be formulated with its own degree of formality (i.e. strictness of applicability criteria), allowing as well for flexible, rigid, and mixed processes. Patterns

may be used both to guide the development process, and to document a development after-the-fact. Patching-in ad-hoc steps in a process is trivial for a pattern-oriented process.

Finally, patterns are scale-invariant, that is, they may be applied on any scale of granularity or abstraction, as long as their application criteria are satisfied. Using a language of process patterns this way, a component-oriented discipline of software construction is easily accommodated: building systems from components results in a hierarchical and self-similar (i.e.: fractal) product structure (cf. e.g. [4, 14, 10, 20]). This is probably *the* single most important advantage process patterns have over traditional processes.

1.3 Outline of approach

The approach pursued in this paper may be characterized as follows: first, I summarize UML's facilities for modeling process, and propose some small extensions. Note that this approach is based on the very recent UML 1.4, which contains some improvements over UML 1.3. Then the descriptive schema laid down in [12] (and very similarly in [6], the other of the two major design patterns books) is adapted to fit the new requirements of the process domain.

1.4 Related work

There is already some work done on pattern languages for software processes. The earliest reference that I know of is the work on "process chunks" by Rolland et al. (cf. [24]). Process chunks are 4-tuples consisting of situation, decision, action, and argument, formalizing design decisions taken during the requirements elicitation. Neither UML nor patterns in the modern sense are used.

Then there is a number of papers like [4, 11, 27] which already use the term process pattern, but remain superficial. There, even less detail is provided to describe the patterns. Neither a general scheme nor interesting individual aspects (e.g. classification, process, participants) are described in any detail.

A more serious approach is presented in Catalysis (cf. [9]), where there are 60 different patterns, each of which is described by three to five aspects (name, intent, strategy, and sometimes considerations and benefits) using natural language and ad hoc sketches. While presenting the first real example of a language of patterns, Also, Catalysis does *not* use UML, except claims of the opposite, and even the notation it does use (which is vaguely reminiscent of UML) is not used in the process patterns. Neither the roles and responsibilities involved, nor the document types required and provided, nor the actual design activities are described in any great detail.

Finally, there are the books by Ambler (see e.g. [2]). They deal with what in the terminology presented here would be organizational, or process phase patterns. What I call process pattern is termed task pattern by Ambler. Also, Ambler does not really use the UML (despite the cover page claim of the opposite): there is a handful of use case diagrams, and that's it. Also, there is no elaborated schema (like we present it here), much less a classification schema.

In fact, Amblers books are really general purpose introductory textbooks on software process (though comparatively good ones).

Compared to all the above mentioned approaches, our approach is the only one which seriously tries to incorporate the UML (and certainly the first one to take into account the UML 1.4). Also, all other mentioned approaches use much less elaborated and precise descriptive schemes for patterns. With the exception of Rolland et al., they don't even present a schema at all.

2 An example

As an example, consider the following family of process patterns, consisting of four process patterns called *Realize*, *Implement*, *Buy/Reuse* and *Refine*. Figure 1 presents an informal sketch of the tasks involved in these process fragments, and the control flow.

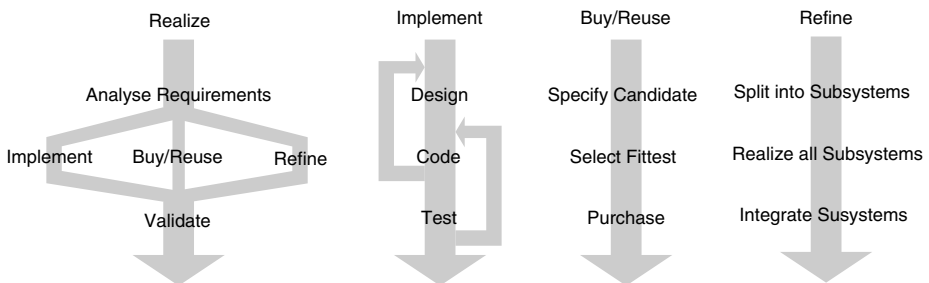


Fig. 1. A simple language of process patterns (informal sketch).

Realize The overall goal is to realize some system. Whichever strategy will be chosen for the realization proper, it is certain that the task will need definition before, and validation afterwards.

Implement In some cases, a system may implemented right away without further fuzzing about. This is the case for very small systems, for a quick-and-dirty prototype, or if there is a lot of experience and adequate support of tools and libraries.

Buy/Reuse Alternatively, one may purchase a prefabricated component from a kind of component market, that is, either buying it from a third party, or reusing it from internal sources.

Refine If a system can be neither implemented directly nor purchased from a third party, one may chose to split it up into smaller subsystems, and try to tackle these individually.

Obviously, applying this set of patterns gives rise to a recursive structure: consider each pattern as a rule in a grammar, and the application as a derivation (cf. Figure 2).

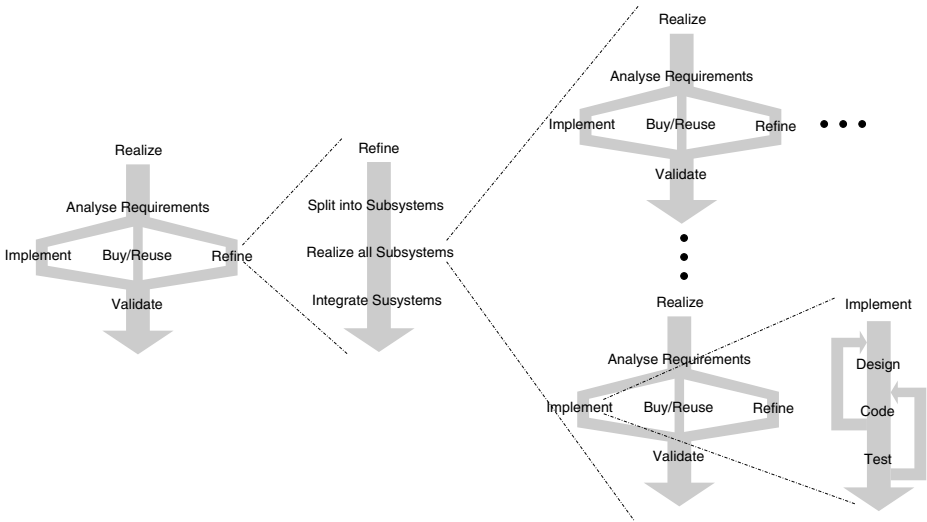


Fig. 2. Applying the pattern language (informal sketch).

Admittedly, this pattern language is somewhat simplistic. A practical example, however, would by far exceed the space limit set for this article.

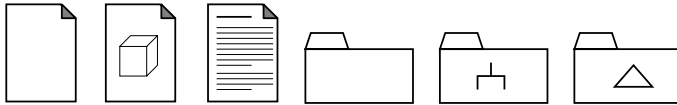
3 Using UML to describe software processes

The UML is *the* most widely understood design notation in software engineering. Obviously, it would be helpful to use it for the description of processes, so as to benefit from the tools, knowledge, and research results concerning UML. Up to version 1.3, the UML contained very little for describing software processes (cf. [16, 1]). The UP did present some notations, but their relationship to the UML metamodel was undefined. Technically, the UP is not part of the standard proper, but it is a kind of quasi-standard in this area. The UML 1.4 now provides the new metaclass **Artifact**¹, with stereotypes like **«document»**, **«file»** and **«executable»** (cf. [21, p. 2-17–2-21]). It also provides a much more detailed standard profile for software process modeling which focuses on the product structures. In the remainder, I will compile a set of notations and concepts (plus additions, where necessary) from UML 1.4 and UP that should be sufficient for most tastes.

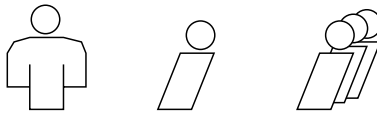
Documents Though the concept has been included in the move from version 1.3 to 1.4, there are no notations presented. I propose to use the folklore symbols (also used in UP), that is, an upright rectangle with a dog's ear for documents. Different types may be distinguished by additional inscriptions or

¹ When referring directly to the metamodel, metaclasses will always be printed in sans-serif font.

icons. UML diagrams should be represented by a new stereotype `<<diagram>>` with a string-valued tagged value `kind` indicating the type of diagram. **Models**, **Subsystems**, and all kinds of **Package** in general should be represented as iconified package-symbols, possibly with further inscriptions or icons attached to differentiate between different kinds. See the following sketch for some examples (left to right): any document, a text, a deployment diagram, any package, a subsystem, a model.

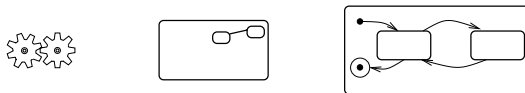


Resources Neither the UML proper nor the standard profiles provide a concepts for resources like tools or people; the notion of **Actor** serves only to represent a neighboring system. The UP, however, distinguishes between *Resources* (roles) and *Workers* (concrete people or tools that are capable of playing these roles). The UP introduces the symbols below for resources, workers, and groups of workers.



Further extensions might become necessary in the future, such as for describing organization structures, but this is beyond the scope of this paper.

Tasks The structure of a task may be described as an activity diagram. The UP introduced a symbol consisting of a pair of cog-wheels for coarse-grained activities. These may be considered as syntactical sugar for **SubactivityStates**. In the UML, it is not legal to present the refinement of a **SubActivityState** by graphical containment (like it is done for **CompoundStates**). This can be quite useful however, so we propose this extension (task, standard **SubActivityState**, refined **SubActivityState**).



Note that, since **Artifact** is a **Classifier**, it may occur in activity diagrams to symbolize the type of an **ObjectFlowState** (see Section 4.1 below). Observe also, that now workflow may be described in coarse and refined versions.

Note that the extensions proposed en passant are conservative and mainly only syntactical, that is, they could be easily included into the conceptual framework of the UML metamodel.

Now, the UP introduces a new kind of diagram. This simple chart shows the artifacts and resources required and produced by a task. It is found throughout the UP, used informally, but not defined properly. With these notations it is

now possible to present e.g. the process fragment Refine from above in greater detail, and with more precision (see Figure 4). Note that we have presented only the control flow, but not the data flow. A more worked example is presented in Section 5.

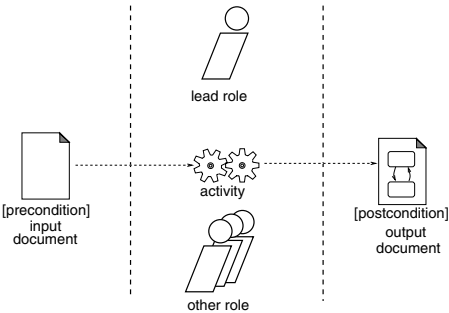


Fig. 3. The UP schema for tasks.

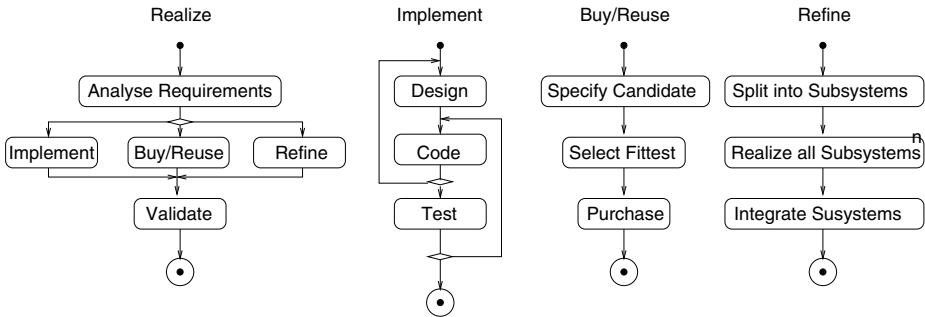


Fig. 4. Presenting tasks by activity diagrams: the language of process fragments of Figure 1 as UML activity graphs. Note the usage of **dynamicMultiplicity** in the Refine-task. We have omitted the data-flow here for simplicity (cf. Fig. 5, right).

4 Describing process patterns

Process patterns can be described by a schema similar to that known from result patterns. The overall descriptive scheme for process patterns and the terminology are taken from [12, p. 5f]. In the following subsections I will first describe the overall schema, and then the modifications resulting from transferring it to the process domain (see [4] for a discussion of the requirements of process vs. result structure patterns).

4.1 Overall description schema

The two best known books on design patterns [6,12] propose very similar schemas for the description of design patterns. In order to increase the acceptance of our approach, I adopt this approach as far as possible. See Table 1 for a comparison of the schema used in [6,12] and here.

POSA [6]	GOF [12]	this paper
Name	Pattern Name	Name
n.a.	Classification	Classification
Problem	Intent	Intent
Also known as	Also known as	Also known as
Example	Motivation	Motivation
Context	Applicability	Applicability
Structure	Structure	Process
Dynamic Aspects	Collaborations	n.a.
n.a.	Participants	Participants
Implications	Consequences	Consequences
Implementation	Implementation	Implementation
Sample Solution	Sample Code	Sample Execution
Applications	Known Uses	Known Uses
References	Related Patterns	Related Patterns

Table 1. Comparison of pattern schemas.

Name First, each pattern has a name. For process patterns, this is usually identical to the name of the task that is supported by this pattern.

Classification As pattern catalogs may grow rather large, there must be a systematic way of retrieving them, which is facilitated by this aspect. This aspect is not present in [6], and appears as part of the name aspect in [12]. See Section 4.1 below for details.

Intent A intuitive account of the rationale of the pattern, its benefits and application area.

Also known as Sometimes, there is not a unique best name of a patterns, but a number of equally adequate names, which are given here.

Motivation A scenario that illustrates the problem, applicability conditions and purpose of the pattern.

Applicability The prerequisites for applying a pattern, the context where it may be applied. In the scope **Capsule**, this may be specified by the required Views and possibly consistency conditions established. includes preconditions

Process In design patterns, this aspect is called “structure” and contains a OMT class diagram. For a process pattern, the equivalent of “structure” is the process, that is the causal structure of activities. See Section 4.1 below.

Participants In result patterns, this aspect refers to the classes involved. In process patterns, the participants are the documents and resources (including people) that play a role in the pattern. See Section 4.1 below.

Consequences Discussion of the advantages and disadvantages of using the pattern, e.g. some quality goal or other end that is achieved by this pattern.

Sample Execution The implementation of a design patterns is a program, that is, an expression of a programming language. The implementation of a process pattern is not a program for some computer, but a procedure in an organization, involving people, tools, organizational facilities and an array of personal and interpersonal techniques. See Section 4.1 below.

Implementation Discussion of implementation-specific problems, i.e. problems related to the realization of the pattern. As process patterns are realized by organizations (see previous aspect), this aspect refers to problems related to organizational and tool problems.

Known Uses Applying a pattern may be easier when guided by concrete examples where the pattern has been applied to general acclaim.

Related Patterns Relationships to other patterns, i.e. alternatives, companions, or prerequisites and discussion of differences.

This schema does not contain the aspect **Collaborations** of [12] (called “Dynamic Aspects” in [6]): it is used to represent the behavior (and dynamic structure) of the Structure. This is already catered for by the new aspect Process, and may be omitted.

Classification aspect Patterns always exist only as part of a *language* of patterns—this is one defining criterion. These pattern languages may become rather large, and so cataloging and retrieval become important practical problems. These problems have been studied in the reuse community (see e.g. [22]). There, the terms *facette* and *simple class* have been coined for the dimensions of classification, and their respective values.

There are several classification schemes for result patterns.² Buschmann et al. propose to use the abstraction level and the problem category ([6, p. 362ff]). The Gang-of-four propose to use the purpose and the scope (see [12, p. 9ff]). For process patterns, these schemes obviously need adaptation, both concerning do not apply. In their stead, I have identified four classification *facettes*: abstraction level, phase, purpose and scope. I shall now explain the simple classes of these *facettes*.

abstraction level In [6], the three abstraction levels of idioms, design patterns, and architectural patterns are distinguished. Analogously, process patterns may be classified as techniques, process patterns proper, and development styles.

phase Design patterns are attributed to a problem category. The analog for process patterns is the development phase (in the classical sense), that is, specification, design, realization, and maintenance. Others are also possible, of course.

² The popular patterns-books do not use the terminology known from *facette*-classification, but the ideas are identical.

purpose In [12], result patterns are classified as “creational”, “structural”, or “behavioral”. For processes, there are other purposes, such as the administration, the construction proper, and quality assurance.

scope Finally, the design patterns may be distinguished according to whether their scope is an object or a class. Since this is a bias towards object-oriented technology which is soundly out of place in the context of processes, these simple classes are replaced by such that refer to the entities occurring in traditional processes, e.g. “architectural style”, “product line architecture”, “reusable component”, “implementation module”, or “test case” may be distinguished.

The simple classes given in each facette may differ from project to project—the ones mentioned here are just plausible defaults. Also, there is usually no hard and fast dividing line between the simple classes of the respective facettes. Note that granularity is not a useful classification criterion for a fractal process.

Process aspect Many people seem to think that the structure aspect (i.e. an OMT class diagram) of a design result pattern *is* the pattern. Process patterns obviously do not have a structure in this sense: they specify a (part of a) process. Consequently, this aspect has been renamed to “process”, and it consists of an UML workflow diagram (typically the refined variant) rather than a static structure diagram.

Instead of activity diagrams, any of the other UML dynamic notations could be used. In fact, *any* other notation for the description of software processes, for example, Petri nets, Rules, activity trees/transaction graphs, or programming language-like notations. None of these, however, reaches the degree of acceptance the UML has (and will have for some time to come).

Participants aspect There are two kinds of participants of a process pattern, artifacts and resources. In the terminology of the UP, resources refers to tools and machines as well as to people. Artifacts may be either prerequisites or deliverables of a process pattern, or both. To represent the participants aspect, the UP’s schema may be used.

Applicability aspect Apart from the traditional contents of these aspects (i.e. natural language descriptions of the applicability conditions of using a pattern), in our approach, this aspects may be refined by formal preconditions on participants. This is particularly relevant for documents, of course, but may also be applied to resources. This opens the road to formal analysis of software processes and automatic enactment.

At this point, a little digression into the UML metamodel is called for. First observe that all notational elements of the UML correspond to a metaclass, i.e. a state chart diagram corresponds to a `StateMachine`, an activity diagram to an `ActivityGraph`, an action state node to an `ActionState`, an object flow node to an `ObjectFlowState` and so on. I have introduced `Document` as a special kind of

Classifier, and thus it may have a `StateMachine` to describe its lifecycle. In the context of `ActivityGraph`, an `ObjectFlowState` has a `type`, that is, an association to a `Classifier` such as a `ClassifierInState` (and `Artifact`). This may be used to represent the current state of the document's lifecycle.

Or, in other words (and ending the digression), documents and other resources have lifecycles, and a particular state of these lifecycles may be used to inscribe document (or resource) icons in workflow diagrams. For instance, artifact may be in states “checked out”, “tested”, or “approved” and so on. So, this state may be used as a specification of a precondition of a resource. Also, any other side conditions (including timing constraints, or complex logical predicates) may be expressed using OCL.

Consequence aspect All that has been said in the previous section applies analogously to the consequences aspect: the state attached to a document or other resource is a postcondition, rather than a precondition.

Sample Execution aspect As I have said before, design result patterns are realized as programs, and process patterns are realized by people in organizations, equipped with tools and applying techniques. So, the process pattern analog of sample code is a sample execution, giving an example of how the respective process pattern might be realized by an organization, that is, who is responsible for what, which techniques and tools are appropriate, how parts of the work are to be synchronized and integrated, which change and version control system may be used and so on.

For example, an initial class model may be generated by conducting structured interviews with domain experts, and then extracting nouns and verbs from such interviews to define analysis level classes and methods, respectively. For analysis level integration testing, role playing with paper prototypes; for user interface design and validation, storyboards are useful, and so on. On the design level, techniques like walk-throughs, inspection, and automatic consistency checking (cf. [26]) might be appropriate.

Related Patterns aspect In design patterns, this may only refer to other design result patterns. Process patterns, however, may refer to any kind of patterns, including result patterns, other process patterns, and organizational patterns (cf. [13, 7]).

5 Using process patterns

Consider now the pattern-representation of the process fragment `Refine`. This example is abstracted to the bare minimum—real life examples would be more elaborated.

Name `Refine`

Classification process pattern / realization / construction / architecture

Intent Realize a subsystem by a divide-and-conquer strategy.

Also known as Top-down, divide-and-conquer.

Motivation When a system is too complex to be implemented directly, and if purchasing it from another source is not possible, one might approach the problem by splitting up the system in simpler subsystems.

Applicability Whenever a complex system is to be realized. Requires the functionality of the system to be specified.

Process See Figure 5, right.

Participants See Figure 5, left. Artifacts used: system specification consisting of use cases, scenarios, other requirements, plans for budget, resources, and schedule. Artifacts created: system implementation consisting of executable, documentation, and validation report. Resources: varying.

Consequences The only realistic choice for complex systems that can not be purchased. Requires that the system in question can be divided into smaller parts at all (not necessarily the case e.g. for time-critical systems). Once the system has been split up, it may be difficult to refactor it. Requires considerable architectural expertise on behalf of the people doing the split.

Sample Execution omitted here due to space restrictions

Implementation Currently no adequate CASE-tool support available.

Known Uses Plenty, e.g. compiler construction.

Related Patterns Usually applied as a consequence of the Realize pattern, which in turn is called again for each of the new subsystems.

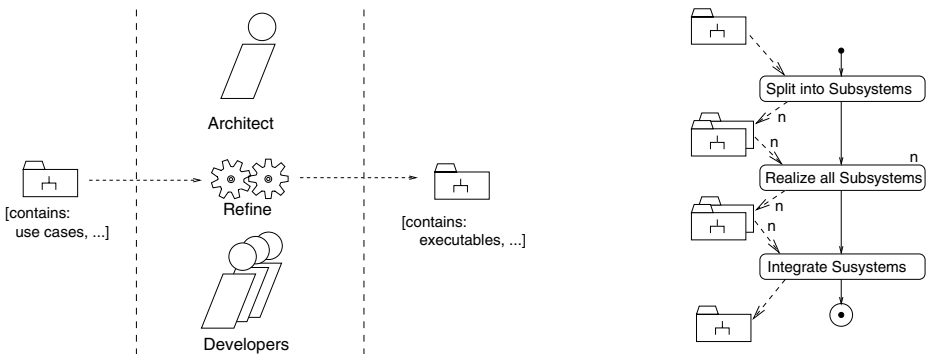


Fig. 5. The Refine pattern: participants (left) and process (right).

Suppose that the language of patterns sketched first in Figure 1 were available in this scheme. There are two fundamentally different ways of applying languages of patterns, namely after-the-fact, to document some result, or generatively to instantiate some process. In the first case, there may be steps in the process that are not documented by patterns, so the developers can focus their effort on those

parts of a project's process that needs their attention most. In the remainder, we consider only the second kind of process pattern application.

Starting from some predefined initial state (that is, a set of artifacts), developers may now apply any of those patterns, whose applicability aspect is satisfied, that is, all participants are available in the required state, and all OCL constraints are satisfied. Observe that applying pattern y on (some of) the artifacts created by pattern x amounts basically to a sequential composition by merging these artifacts (cf. Figure 6).

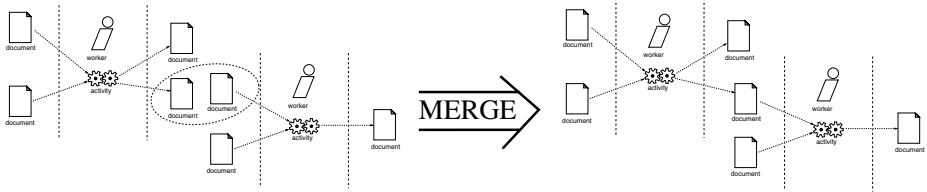


Fig. 6. Individual patterns may be merged to form a development graph.

This also shows, how a traditional process may be re-constructed using a suitable language of process patterns. Some of the benefits of pattern languages for processes may be realized even here, e.g. a more compact representation, usage of UML, and better control over where to demand which level of formality.

Both activity and document-oriented process descriptions may be achieved using process patterns: to achieve an activity-oriented style of process description, the document-types and states are simply left abstract. To mimic a document-oriented style, the documents are filled in with all necessary detail, but the activities are left coarse.

In a rigid development process, both of these aspects would be fully specified. Only process patterns provided with the process could be used to create or modify documents. In a looser development process, the developer would also be allowed to create and modify documents in an ad hoc style. Later on, these activities might again be cast into the framework of the process patterns language, should this be desired, as a form of commenting.

6 Conclusions

Summary In this paper, I have defined a description scheme for process patterns very similar to the one used for design result patterns. Within this scheme the UML is used to describe the aspects process, participants, applicability, and consequences.

In a nutshell, process patterns may be used to describe processes at an arbitrary level of precision. They are now generally understood, as is the UML. The

kind of process patterns I have presented here can easily accommodate traditional and component-based (i.e. fractal) process models. In general, they offer a very flexible, yet precise and generally understood description formalism.

Other approaches are much less elaborated, and, by not using the UML, also of smaller practical use.

Future work Our group is currently working on a CASE-system to support usage of process patterns. In particular, cataloging and retrieval, enactment, and checking of satisfaction of abstract design-guidelines will be supported in this tool. With this tool, it will then be possible to conduct field studies of realistic size. So far, I have use the kinds of process patterns proposed here only in the classroom, though with encouraging results. In [26], I have proposed a process language for architectural modeling, but other application areas, including those of classical workflows are possible, but have not yet been examined.

One other important other strand of our work is the definition of a formal semantics for UML **ActivityGraphs**. This will be necessary to enact and formally analyze workflow graphs. As of writing this, there is no generally agreed upon semantics available. However, there is a plethora of more or less complete formal semantics for UML **StateMachines**, and **ActivityGraph** is a much simplified case of these. Recent advances in this area (cf. [5, 3]) and our own work should amount to a satisfying formal semantics pretty soon now. Such analysis methods would apply as well to process patterns as to traditional process descriptions.

References

1. Thomas Allweyer and Peter Loos. Process Orientation in UML through Integration of Event-Driven Process Chains. In Pierre-Alain Muller and Jean Bézivin, editors, *International Workshop «UML» '98: Beyond the Notation*, pages 183–193. Ecole Supérieure des Sciences Appliquées pour l'Ingénieur—Mulhouse, Université de Haute-Alsace, 1998.
2. Scott W. Ambler. *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. Cambridge University Press, 1998.
3. Alistair Barros, Keith Duddy, Michael Lawley, Zoran Milosevic, Kerry Raymond, and Andrew Wood. Processes, Roles and Events: UML Concepts for Enterprise Architecture. In Selic et al. [25], pages 62–77.
4. Klaus Bergner, Andreas Rausch, Marc Sihling, and Alexander Vilbig. A Componentware Development Methodology based on Process Patterns. In Joseph Yoder, editor, *Proc. 5th Annual Conf. on the Pattern Languages of Programs (PLoP)*, 1998.
5. Christie Bolton and Jim Davies. On Giving a Behavioural Semantics to Activity Graphs. In Reggio et al. [23], pages 17–22. Also appeared as Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Fakultät für Informatik, October 2000.
6. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture. A System of Patterns*. John Wiley & Sons Ltd., 1998.

7. James O. Coplien. A Generative Development-Process Pattern. In Coplien and Schmidt [8], pages 183–238.
8. James O. Coplien and Douglas C. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, 1995.
9. Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks with UML. The Catalysis Approach*. Addison-Wesley, 1999.
10. Brian Foote. A Fractal Model of the Lifecycle of Reusable Objects. In James O. Coplien, Russel Winder, and Susan Hutz, editors, *OOPLSA'93 Workshop on Process Standards and Iteration*, 1993.
11. Brian Foote and William F. Opdyke. Lifecycle and Refactoring Patterns That Support Evolution and Reuse. In Coplien and Schmidt [8], pages 239–258.
12. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
13. Neil B. Harrison. Organizational Patterns for Teams. Monticello, Illinois, 1995.
14. Wolfgang Hesse. From WOON to EOS: New development methods require a new software process model. In A. Smolyaninov and A. Shestiatyynow, editors, *Proc. 1st and 2nd Intl. Ws. on OO Technology (WOON'96/WOON'97)*, pages 88–101, 1997.
15. Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
16. Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel. Using UML for Software Process Modeling. Number 1687 in LNCS, pages 91–108, 1998.
17. Marc I. Keller and H. Dieter Rombach. Comparison of Software Process Descriptions. pages 7–18, Hakodate, Japan, October 1990. IEEE Computer Society Press.
18. Alexander Knapp. *A Formal Approach to Object-Oriented Software Engineering*. PhD thesis, LMU München, Institut für Informatik, May 2000.
19. Ralf Kneuper. Requirements on Software Process Technology from the Viewpoint of Commercial Software Development. Number 1487 in LNCS, pages 111–115. Springer Verlag, 1998.
20. Meir M. Lehman. Programs, life cycles, and laws of software evolution. *IEEE Transactions on Software Engineering*, 68(9), September 1980.
21. OMG Unified Modeling Language Specification (draft, version 1.4). Technical report, Object Management Group, February 2001. Available at <http://cgi.omg.org/cgi-bin/doc?ad/01-02-14>.
22. Ruben Prieto-Diaz. *Classification of Reusable Modules*, volume I - Concepts and Models, pages 99–124. ACM Press, 1989.
23. Gianna Reggio, Alexander Knapp, Bernhard Rumpe, Bran Selic, and Roel Wieringa, editors. *Dynamic Behavior in UML Models: Semantic Questions. Workshop Proceedings*, Oktober 2000. Also appeared as Technical Report No. 0006 of the Ludwig-Maximilians-Universität, München, Fakultät für Informatik, October 2000.
24. Colette Rolland and Naveen Prakash. Reusable Process Chunks. Number 720 in LNCS, pages 655–666. Springer Verlag, 1993.
25. Bran Selic, Stuart Kent, and Andy Evans, editors. *Proc. 3rd Intl. Conf. <<UML>> 2000—Advancing the Standard*, number 1939 in LNCS. Springer Verlag, October 2000.
26. Harald Störrle. *Models of Software Architecture. Design and Analysis with UML and Petri-nets*. PhD thesis, LMU München, Institut für Informatik, December 2000. In print, ISBN 3-8311-1330-0.
27. Bruce Whitenack. RAPPeL: A Requirements-Analysis-Process Pattern Language. In Coplien and Schmidt [8], pages 259–292.

Extending the Software Process Culture – An Approach Based on Groupware and Workflow

Renata Araujo¹ξ and Marcos Borges²

¹Departamento de Ciências da Computação, UFRJ, Rio de Janeiro, RJ, Brasil
renata@nce.ufrj.br

²Instituto de Matemática/Núcleo de Computação Eletrônica, UFRJ, Rio de Janeiro, RJ, Brasil
mborges@nce.ufrj.br

Abstract. This work proposes the use of groupware technology as an element for extending software process culture within development teams. The proposal relies on the application of workflow systems for software process support and on awareness mechanisms for software process visualization and understanding. We argue that this awareness information may help participants to both understand the processes they execute, and to better accept the idea of defining, standardizing and continuously improve their work tasks. We built an environment – PIEnvironment – using a commercial workflow system and evaluated its use for the enactment of some software process activities.

1 Introduction

Most than ever, organizations have been facing the challenge of improving the quality of their working processes as a strategy to remain alive and competitive. Many companies are struggling to reengineer, automate and improve the way they perform their business. In the software development area, this movement has been expressed by the growing research in software process support and by the wave of software assessment and improvement [1].

In spite of all this effort, reality has shown that a high number of organizations dealing with software development remain immature. Techniques for process modeling, assessment and improvement are not adopted, in many cases even unknown, and may require a great deal of effort to be carried out [2][3].

To introduce the idea of defined and formalized work processes in an immature organization means to change the way of working, to make people follow guidelines, to track and measure work performance and to knowledgeably distribute tasks among developers. Therefore, the successful introduction of new work practices into an organization requires the establishment of a positive culture towards the idea of having a defined process and the construction of an infrastructure for its support [1].

Our work was conceived as an agent for helping workers in creating this new culture. We suggest the use of technologies with the potential of reducing the distance

ξ The work reported in this paper was conducted while the first author was affiliated to the Computing and Systems Engineering Programme at COPPE/UFRJ.

between work teams and their working processes, as well as supporting them in the execution of their tasks. Groupware [4] and, in particular, workflow [5] are the technologies we use to support software process.

In our view, collaborative technology offers resources which allow process participants to have an extended notion and visualization of their working practices. One of the aims of groupware technology is to make collaborative processes more explicit to their users by offering *awareness resources* [20]. Awareness resources provide groupware users with information that helps them to understand what is happening during group work, and, consequently, giving context to their individual activities. We argue that an extended notion and visualization of working processes through awareness resources can make workers more committed to the improvement of their practices, to continuously learn about their own process and to have a balance between autonomy and responsibility in process execution.

The next section outlines the main aspects related to changing software process culture in organizations. In Section 3, we discuss how improving collaboration and process awareness can help to introduce the idea of process definition and improvement in organizations. In Section 4, we present the PIEnvironment, a software process support environment based on workflow technology, built in order to evaluate our proposal. In Section 5 we describe the design, execution and results of two case studies using the PIEnvironment. In Section 6 we conclude the paper.

2 The Problem: Changing Software Process Culture

One of the first steps for the establishment of a process culture within an organization is the selection of the software engineering best practices that will bring improvements in the organizations process development performance and quality. Guiding organizations in choosing the practices that best fit their needs has been the aim of the process reference models – CMM, ISO, SPICE, etc [6].

Process culture also involves conducting an overall approach for continuously introducing the selected best practices and to make people aware of their needs and benefits. These have been the aims of the proposed process improvement approaches such as IDEAL, QIP, AINSI and so on [7][8].

The establishment of a process culture also implies the appropriate support to the execution of the chosen best practices. As a result, the area of software process automation has dedicated a great amount of work in developing environments and tools to support software engineering workers [9].

However, in spite of the model or approach adopted, many of the possible risks that can happen in an improvement plan are directly related to the acceptability of the initiative. Along with the implementation of an improvement initiative in an organization, changes are expected to occur not only in the process but also in team members' attitudes, values and behavior. Unfortunately, practice usually shows us that cultural issues are often overlooked, and improvement initiatives are clearly imposed to workers by higher management.

3 The Approach: Improving Collaboration and Awareness

To provide developers with continuous consciousness and knowledge about their processes is an effort that includes more than the decision to adopt or not certain reference model. It involves cultural issues related to: values, commitment and participation. Our approach is to provide means and tools aiming at promoting the introduction of this culture among software developers. Among the issues that comprise culture change, our approach relies on three main aspects – collaboration, awareness and participation.– as detailed in the following sections.

3.1 Collaborative Technology – Agent for Process Improvement

Groupware technology aims to assist group members in: communicating, sharing information, resources and tasks, and coordinating their responsibilities and activities through the use of networks and computers [23][26]. The research in collaborative technology and the investments of groupware industry rely on the potential of this technology for changing and improving group work. Since today's business is strongly dependent on communication and interaction between inter and intra organization members, groupware technology has often been associated with work reengineering initiatives and the new way of conducting global business [4]. Hence, there is no doubt that collaborative support may improve work quality, although there is still the question of how this improvement can be asserted according to each work environment and context.

The software development area has also been facing the challenge of “reengineering” the way people work. The research in software process automation and support has already recognized the importance of the collaboration aspect in software processes and the need to support this collaboration accordingly [10][11].

We could state that improving software processes also means improving the collaboration that exists among development teams. Therefore, to encourage, to support and to enforce collaboration becomes an essential aspect of software process improvement.

Workflow Management Systems. One kind of groupware application – workflow management systems (WFMSs) [4][24] – emerges as a potential infrastructure for supporting software processes [11][12]. Although WFMSs have their origins in the business area, they can be considered as an enabling technology for process automation in general, and as software process centered environments.

WFMSs main functionalities comprise process modeling and enactment. Basically, they offer resources for modeling processes based on activities, performers and products. From the definition of a process, the WFMS controls its enactment by following the defined flow of activities in the process and delivering the execution of each activity to the assigned participant at the right moment. At any time, each participant may have access to his “work list” where he can find all the tasks that are under his responsibility at the moment, attached with instructions about how to

proceed in order to complete them. Once the participant completes a task, the WFMS engine proceeds to the next step of the defined process flow.

As far as this work is concerned, what attracts us in WFMS is its collaborative nature. The concept behind WFMSs is to integrate users (process participants) to the process itself, and to distribute work responsibilities within the team according to the defined process. This way, WFMSs helps participants to become aware of their responsibilities instead of strictly controlling their work.

Moreover, the new generation of WFMSs relies strongly on open and interoperable architectures as well as on the WWW, making these systems highly flexible and accessible. High accessibility is a strong feature when we are concerned about integrating people inside an organization as well as when we are interested in bringing people “closer” to their processes.

3.2 Awareness Information and Organizational Learning

Software organizations spend efforts in improving their processes as an attempt to achieve a higher level of *maturity* in their production. One of the elements for achieving maturity is the ability to learn. Organizations in higher maturity levels are those that already developed a culture where people know about their processes, and where exists an infrastructure that allows existing processes to be learned.

Following this point of view, our proposal focuses on how to provide development teams with resources for *being aware* of the process they execute. These awareness resources have a fundamental role in helping people to recognize and learn about the way they actually work, as well as to recognize problems and improvement possibilities. *Awareness* is the concept through which we will try to promote the establishment of an ideal process culture, helping organizations to introduce and accept the idea of process definition and formalization.

Awareness is also an important aspect in collaborative applications [20]. Collaborative tools must provide group members with information for making them aware of the interaction with each other. It is argued that awareness information is essential for improving collaboration. One cannot work collaboratively without knowing about the activities performed by others and the overall group work. So, we argue that we should provide developers with awareness information in order to improve collaboration.

Awareness of Working Processes. According to Stenmark [25], knowledge can be classified as tacit knowledge when it “resides within us, and manifests itself through our actions, and we therefore need not to document it for our own sake. We just use it.” When we decide to make this tacit knowledge available to someone else to use, we have to, metaphorically speaking, put it in “words”, that means, to codify it and make it explicit. This way, the first step to make an organization learn is to turn the tacit knowledge that exists in each department or employee’s mind to its explicit state, so that other sectors or employees can be aware and learn about it.

To induce teams to accept the idea of process traceability and improvement, it is necessary to explicitly introduce the concept of a process being an important element and to create resources that will aid teams to perceive the process in which they are involved.

Many studies suggest that working processes are important components for promoting organization learning. Following this perspective, process automation technology has been considered as an element for helping workers in learning about organizational processes. By providing resources for process definition, modeling and enactment, process technology can be useful to show and explain how work is carried out [15][16][17].

In our approach, workflow management systems will be the enabling technology for supporting software process and the infrastructure for process awareness and learning.

Awareness of Collaboration. Software processes are usually visualized from the point of view of the set of activities and practices that comprise them. Nevertheless, one neglected aspect in software process definition and support is its collaborative dimension. A research conducted in [22] showed that software developers spent, in average, more than a half of their working time interacting with others. Their work also reported that developers often needed to apply efforts to find out whom they should contact to for solving problems.

Unless the development group is very small, process actors often don't have an explicit perception of how they contribute to each other. In many cases, participants may have a clear idea of their individual responsibilities, according to their roles: "I am a programmer and I codify classes". Sometimes, they may have the idea of being part of a team and of other participants' responsibilities: "I'm a programmer and I'll codify the classes brought by the analysts and they will be tested by someone else". Very often, knowledge about collaboration is restricted to "who shares the same artifact with me" and not in respect to: "who I need to interact to in order to have my work done" or "who will benefit from my work" or even "how does the whole team collaborate". This lack of knowledge about collaboration reduces process learning and may bring barriers to their execution, acceptance and improvement.

Some studies focusing on the social aspect of software processes have been reported, reinforcing the need for making participants aware of the collaboration that occurs among them [14][19]. One of these studies, directly related to our work, was conducted by CAIN and COPLIEN [13][14]. They proposed an approach that focuses on modeling processes using roles and their relationship during process execution.

They used CRC cards to make employees write down their roles, responsibilities and interactions with other roles in an organization. Cards are assigned to each role (actor), its responsibilities are enumerated and their relationships (collaboration) with other cards identified. The technique is highly participative in its nature, involving all actors in collecting data about interactions that occur during the process. As a result, a graph can be generated using the information provided by actors where each node of this graph is a card (an actor) and the arcs are the collected interactions among them.

As shown by Cain e Coplien's studies, the notion about the overall interaction between actors, offers valuable information on how collaboration is viewed in the organization. The great majority of process participants felt surprised to verify that the

preception they had about the way the process was executed was very different from the resulting model being presented. They also reported that this technique made participants modify the way they saw their work.

3.3 Participation

Another consequence of achieving process maturity is the growth of participants' responsibility with respect to actions and decisions. It seems that immature organizations need an effective and present management, while mature organizations have more self-directed teams [16]. As organizations achieve higher maturity levels, they have better consciousness about the processes they execute and their teams can better direct their activities based on the process defined for the organization, since it is already familiar to the majority of employees.

When more aware of their processes, workers can easily identify problems and inconsistencies, stimulating continuous improvement. This process visibility, when added to the possibility for communicating their impressions and observations, may produce additional satisfaction, commitment and responsibility with their work [21].

4 An Implementation – The PIEnvironment

We visualized an environment for software process support – Process Improvement Environment – that focuses on the aspects mentioned above. It induces collaboration, provides mechanisms for making software process explicit for the development team, and supplies a channel where participants may register comments and suggestions.

4.1 Awareness of Work Processes – The Role of Workflow

Workflow systems provide awareness resources that help participants understand process definition, the responsibilities of each participant, as well as to allow them to be aware and trace the process they take part in. These features are the starting point for supporting the “dialogue” between processes and performers and are the core of our proposal for providing process awareness

PIEnvironment has as its main core a commercial workflow system [19] in which a software process can be defined, implemented and executed. Through the awareness resources provided by the workflow system, PIEnvironment users can obtain information about their and others' worklists containing the activities (responsibilities) they have with the process at an specific moment; a list of process instances being enacted; the activities comprised by the process and its execution flow (Fig. 1); the status of each of those instances (Fig. 1) and the documents used along the process execution. The system also provides information and statistics about process performance.

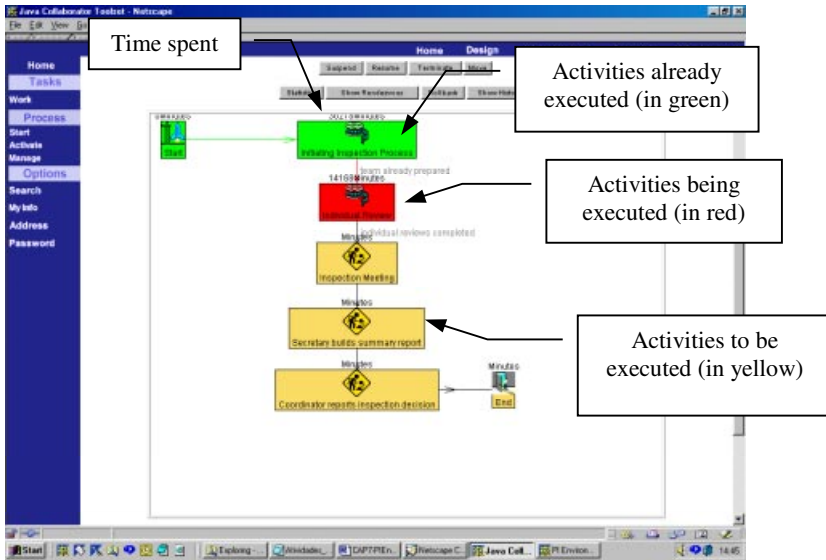


Fig. 1. Process map and status of execution of a process instance

4.2 Awareness of Collaboration – Groups, Roles, and Interactions

The workflow system provides the main resources for viewing process from the perspective of its activities composition. However, the existing collaboration is still difficult to recognize, since most workflow systems don't explicitly handle this issue. For instance, when a group is assigned to perform an activity, each member will only be aware of the necessary interaction with other group members when he actually performs the activity.

We extended the workflow system by developing additional awareness resources. The information provided by these resources includes the awareness of group composition, an overview of individual and group responsibilities and an idea about the existent interactions in the process.

Groups, roles and responsibilities. The first step to understand the possibilities for collaboration along the process execution is to provide participants with knowledge about the team that he belongs to and to recognize each member's role and responsibility. Participants should be able to recognize the groups and roles assigned to contribute to the process, and what activities they perform.

As shown in Fig. 2, for each defined process, the PIEnvironment displays information about process participants and a list containing all process activities. Process performers can be either individuals or groups, so the PIEnvironment allows the visualization of the components of each group that participates in the process. This composition is presented as a tree where one can see the defined roles within the

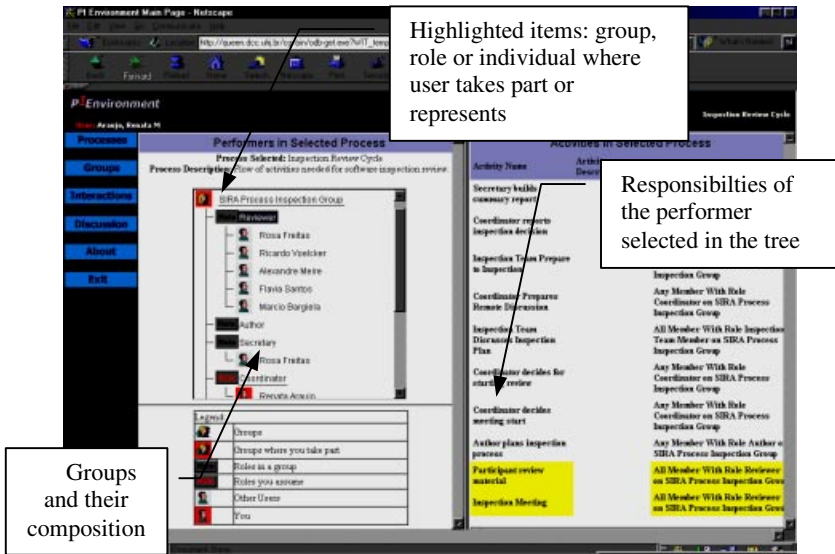


Fig. 2. Group composition and user participation in groups

group and the individuals assigned to each role. From this tree, the user can also be aware of his participation in each group (Fig. 2).

One can view the activities in the process that are under the responsibility of each tree node (groups, roles or individuals). When a tree node is selected, the activities executed by the performer represented by the node are highlighted (Fig. 2).

Interactions. To provide participants with perception about team collaboration, we adopted a solution based on Cain and Coplien's approach for process modeling, mentioned earlier in this paper. This approach focuses on modeling processes using roles and their relationship during process execution.

In PIEnvironment, an interaction graph is constructed by following the process definition implemented in the workflow system. Each node on the graph represents a performer (a group, a group role or an individual). The distinction between each kind of performer (group, role or individual) is obtained by using different colors (green, yellow and red, respectively) (Fig. 3). The user can also visualize his participation in the interaction graph. The roles and groups where the user takes part are highlighted in the graph, as well as the nodes where he appears as an individual.

The arcs on the graph represent the interactions and/or collaboration that may exist among process participants. The interactions on the graph are also collected from the defined process in the workflow system. One criteria used to build the graph is related to the definition of the performer of each activity. If an activity in the process was assigned to a group, the graph displays the possibility of a potential collaboration among the roles which constitute this group.

The PIEnvironment also considers that performers of subsequent activities in the process flow of execution have some degree of collaboration. We assume that if one performer has to wait the execution of a previous activity in order to execute his own activity, that may be an indicator that the results/products of the preceeding performer

are needed for his work. In other words, performers of contiguous activities “interact” through the results/products of his activities and this interaction may be represented in the graph.

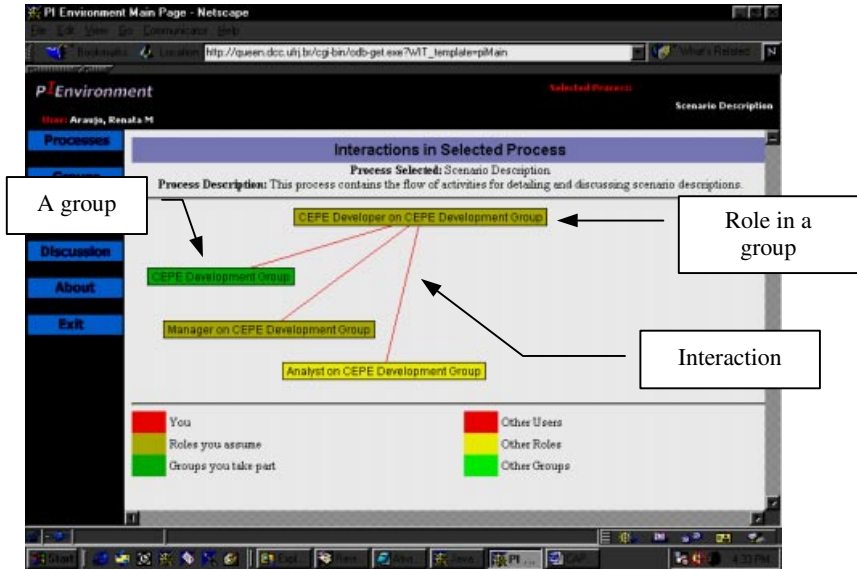


Fig. 3. Example of an interaction graph

4.3 Participation

The PIEnvironment provides participants with a channel for input and discussion of ideas and improvement suggestions for the process. For each process, the PIEnvironment provides a discussion forum where participants can bring up topics for discussion and the others may comment on them.

We believe that the participants’ contributions must be publicized for the whole team. By making these comments public and allowing discussion, we may extend improvement possibilities, as well as the team satisfaction and commitment.

5 Some Results: Case Studies

To verify the augmentation of process culture inside an organization using PIEnvironment, it would be necessary to use our proposal extensively in many development contexts for a long period of time. As a starting point for this verification, we have concentrated on observing the potential of our proposal in

providing awareness, knowledge and consciousness to process participants about their work, and whether they felt satisfied and in favor of the idea of using a defined process.

So, the hypothesis of this work can be stated as: the awareness of work processes helps participants in understanding the process they execute, allowing them to identify improvement possibilities, and, together with the possibility of registering comments about the process execution, participants become more satisfied and in favor of the idea of using a defined process to guide their work.

To start studying these hypotheses, we conducted two case studies, which design and main results will be described in the next sections.

5.1 Design

The questions, which we wanted to find answers for in the case studies were:

Q1 – Do the awareness resources provided by PIEnvironment help participants in establishing a notion about the process they execute?

Q2 – Do the awareness resources provided by PIEnvironment allow participants to identify process improvement possibilities?

Q3 – Does process understanding induce the acceptance of process definition by process performers?

To analyze these questions we defined a set of variables to be measured throughout the case studies execution. These variables are, respectively, *the level of process comprehension* by participants, *the number of contributions to the processes* registered by participants and *the level of acceptance of process formalization*.

These variables can be modified according to participants' previous experience and culture. The ability to understand the process can be greater to those participants who have already had previous experience in using defined processes or who had already accomplished the software engineering practices (scenario definition and software inspections) defined in each of the case studies.

The existence of an established process culture in the organization can also be a facilitating factor for helping participants to accept the use of defined processes. Therefore, we considered the preexistent *process culture* of each participant as an independent variable in our evaluation.

Measurement instrument. We defined a questionnaire to be filled out by each participant after the execution of the case study. This questionnaire contained several questions to evaluate quantitatively and qualitatively the variables outlined above.

In the questionnaire, we measured each participant's process culture by asking about their areas of interest and work, previous experience with software development, knowledge and use of existent reference process models (CMM, ISO etc) and their personal impression and feelings about previous experiences in using defined processes to guide their work.

To measure the level of process understanding, we formulated questions where participants could tell how confident they were in describing process objectives, activities, products, existent collaboration and their own responsibilities.

The number of contributions was measured by counting the topics and comments registered in the PEnvironment discussion forum in addition to specific items in the questionnaire. We asked participants to enumerate suggestions for process improvement and indicated the relevance of their suggestions.

Finally, to measure the level of acceptance of process formalization we asked participants their level of satisfaction with the process enactment and if they were encouraged with the idea of having their processes formalized in the future.

5.2 Application

We conducted two case studies. In each, a group executed software development tasks using the PEnvironment. We modeled two processes to guide each case study execution based on suggested software development practices in the literature, adapted to include some activities/aspects to encourage collaboration throughout the process.

In the first case study, the group consisted of five members, comprising postgraduate students and researchers in software engineering and groupware. They used the PEnvironment for enacting a process for specifying the requirements of a new version of a collaborative application. The process includes activities for conducting the system requirements specification based on scenarios description.

Four (4) postgraduate students made up the second case study group. Some of them also worked on software development organizations. The aim of this group was to execute an inspection process.

Enforcing collaboration, participation and commitment. Collaboration was an aspect that we wanted to stress in the process. Hence, we tried to encourage and explicitly introduce the collaborative aspect into the process models. We defined many activities that involved discussions and decision-making by the team and offered some tools for supporting their accomplishment (for instance, discussion forums in Lotus Notes).

Interesting observations were obtained when we induced team participation and commitment in process modeling. Before being executed, each process model was presented to the team in a meeting. In this meeting participants were able to discuss process objectives and suggest changes to the initial process model.

In both case studies we observed that after the meeting, the final process model was richer and more detailed. Additionally, it covered a series of issues that weren't covered by the initial model.

The changes suggested by the group improved the process in many ways. First, by making it more suitable and attractive for the team, based on the participant's experience and culture. For instance, in the first case study, participants did not have any experience with the kind of activity being proposed (software scenarios definition). During discussions, we noticed that this lack of experience needed to be reflected in the final process by defining process activities in more detail. Second, it was an opportunity for participants to take a first look at the process, to discuss its definition, to clear up some doubts and to present suggestions.

We believe that this experience provided the team better knowledge about their work, as well as greater commitment and acceptance to its execution. Actually, we believe it was an important part of success in making participants more committed to their work. Participants were asked to contribute to the process that they would later perform. Making evident that they are having an important role in the definition of their future tasks made them feel more responsible and committed to process performance and success.

5.3 Results

From the data and observations collected, we were able to answer our preceding questions in the following way:

Q1 – Do the awareness resources provided by PIEnvironment help participants in establishing a notion about the process they execute?

We observed that almost all participants had a high degree of process understanding. We asked participants to describe how they saw the process they executed and in most cases it corresponded with the actual process. Participants were able to clearly describe activities, products, group composition and interactions. This led us to some initial evidence that the environment support as well as the initiative to previously discuss the process definition helped participants recognize their process and the existing collaboration.

Q2 – Do the awareness resources provided by PIEnvironment allow participants to identify process improvement possibilities?

The case studies showed us that the degree of participation with improvement suggestions was low. This result may have many reasons. First, participants reported satisfaction with the quality of the process they performed, what could limit the number of suggestions for improvement. Also, the time spent using the environment was short, since the processes themselves were simple and executed in a short period of time. This may have constrained participants in having enough time to outline eventual problems in process enactment. Also, one fact that contributed to the small number of comments on the process was the fact of having an initial meeting before process enactment. This discussion aided participants to feel more intimate and committed to their processes. However, among the reported suggestions, many of them were highly relevant for improving the process.

Q3 – Does process understanding induce the acceptance of process definition by its performers?

We could observe that there were initial evidences that the environment use, the understanding of the process and the awareness of collaboration facilitated participants to accept the idea of using formally defined processes to guide their work. These evidences are reinforced by the fact that the participants in both case studies had neither the habit of nor the experience in following defined processes but yet, they reported a high degree of process acceptance.

An important issue was the stimulus to collaboration. We included in the processes activities to encourage participant interaction - establishing meetings (some with automated support) and specifying the execution of collaborative activities.

Participants reported a high level of collaboration in their activities and were satisfied with its execution, feeling involved and engaged in the process.

An overview of the main results of the case studies can be summarized in Table 1.

Table 1. Case study results (DC – degree of process understanding, NC- number of improvement contributions, DA- degree of process formalization acceptance; H – high, M – medium, L – low)

		DC NC DA			Previous experience with sw development			Previous experience with the use of a defined process			Organization process culture		
					No	S	H	No	S	H	No	S	H
						O	I		O	I		O	I
						M	G		M	G		M	G
						E	H		E	H		E	H
CASE 1	P1	H	M	M									
	P2	M	L	H									
	P3	H	H	M									
	P4	H	H	H									
	P5	H	H	H									
CASE 2	P1	H	M	H									
	P2	H	M	H									
	P3	H	L	M									
	P4	M	M	M									

5.4 Limitations

The case studies lasted a short period of time. They took approximately two months and participants were not exclusively allocated to that activity.

Almost all participants had previous knowledge about the basic workflow concepts but they were not formally trained on the use of the PIEnvironment. Although the processes used in each case study were not complex, technology barriers could have prevented participants to execute and consequently understand their processes.

The case studies were not conducted within a process improvement initiative. The teams used the environment for executing their activities without being involved in a “movement for change”. We believe that within an initiative for improvement, participants would report different feelings as well as a greater number of suggestions.

6 Conclusion

This work proposes the use of groupware technology for supporting teams and their working processes, aiming at increasing their culture in respect to process definition, use and improvement. We discussed how supporting, inducing and making collaboration explicit may help participants to understand the way they work, to relate themselves to other members in the team, and to better accept the idea of using and continuously improving their working processes. We particularly focused on WFMSs for supporting software process execution and on awareness computational mechanisms as resources for making process explicit for its users. An environment – PIEnvironment – was built and put into experimentation.

Our case studies are a first attempt to validate our hypotheses. They were conducted without rigorous and formal variables manipulation. Our main objectives were to observe the first impact of PIEnvironment in use and to collect some data which could guide us in new observations and in the design of controlled experiments in the future.

The results obtained could not lead us to the conclusion that the PIEnvironment changes software process culture in organizations. But they were very interesting in order to observe that teams that had only some kind of “theoretical” process culture and that had almost no previous experience with software process definition, use and improvement, however reported satisfaction, credit and acceptance on this initiative.

Although acceptance and agreement with the use of defined processes do not guarantee commitment to process improvement, to overcome this obstacle is of great importance to process improvement initiatives, mainly in immature organizations. Hence, we believe that the PIEnvironment can be even more enhanced and used as helper for process learning and acceptance.

For future work, we intend to conduct additional case studies to collect additional data and evidence of the benefits of our proposal. We would also like to analyze how our approach impacts the team productivity and the product quality. Finally, we intend to suggest new mechanisms that can be used to increase participants’ awareness while being supported by workflow systems.

7 Acknowledgements

The authors would like to thank FAPERJ (process #E-26/152.176/2000) and CNPq for financial support, CA and TDI-USA for Jasmine and WebDeploy demo versions, and Coppe-UFRJ, where this work was conducted.

References

1. Zahran, S.: Software Process Improvement – Practical Guidelines for Business Success, 1 ed., Addison-Wesley (1998).
2. Process Maturity Profile of the Software Community - 2001 Year End Update <http://www.sei.cmu.edu/sema/profile.html>, accessed on March (2001).
3. Goldenson, D. R., Herbsleb, J.D.: After the Appraisal: A Systematic Survey of Process Improvement, its Benefits, and Factors that Influence Success. CMU/SEI-95-TR-009 ESC-TR-95-009, SEI, Carnegie Mellon University, Pittsburgh, Pennsylvania (1995).

4. Coleman, D. Khanna, R.: Groupware Technology and Applications. 1 ed. Upper Saddle River, NJ, USA, Prentice Hall (1995).
5. Chaffey, D.: Groupware, Workflow and Intranets – Reengineering the Enterprise with Collaborative Software. 1ed. Digital Press (1998).
6. Wang, Y., King, G., Dorling, A., Wickberg, H.: A Unified Framework for the Software Engineering Process System Standards and Models. In: 4th IEEE International Software Engineering Standards Symposium and Forum, Curitiba, Brasil, pp. 132-141, May (1999).
7. The IDEALSM Model, www.sei.cmu.edu/ideal/ideal.html, last access on March (2000).
8. Briand, L., El Eman, K. e Melo, W.L.: An Inductive Method for Software Process Improvement: Concrete Steps and Guidelines. In: EL EMAN, K., MADHAVJI, N.H. (eds), Elements of Software Process Assessment & Improvement, 1 ed., capítulo 7, Los Alamitos, California, USA, IEEE Computer Society (1999).
9. Christie, A.M.: Software Process Automation. 1ed. Springer-Verlag (1995).
10. Araujo, R.M., Dias, M.S., Borges, M.R.S.: A Framework for the Classification of Computer Supported Collaborative Design Approaches. In: Third CYTED-RITOS International Workshop on Groupware (CRIWG'97), pp.91-100, Madrid, Spain, October (1997).
11. Fuggetta, A.: Software Process: A RoadMap. In: 22nd International Conference on Software Engineering, Limerick, Ireland, pp. 27-34 June (2000).
12. International Process Technology Workshop (IPTW'99), France, September (1999)
13. Cain, B. G., Coplien, J.O.: Social patterns in productive software development organizations, Annals of Software Engineering, vol.2, pp. 259-286 (1996).
14. Cain, B.G., Coplien, J.O.: A Role-Based Empirical Process Modeling Environment. In: Proc. of the 2nd Int Conference on the Software Process, Berlin, Germany, February (1993).
15. Ellmer, E.: Improving Software Processes. In: Proceedings of Software Engineering Environments, Noordwijkerhout, The Netherlands, pp. 74-83, April (1995).
16. Ellmer, E.: A Learning Perspective on Software Process Technology. Software Engineering Notes, vol. 23, n. 4, pp.65-69, July (1998).
17. Becattini, F., Di Nitto, E., Fuggetta, A., Valetto, G.: Exploiting MOOs to Provide Multiple Views for Software Process Support. In: International Process Technology Workshop (IPTW'99), Villars de Lans, France, September (1999).
18. Christie, A. M., Staley, M.J.: Organizational and Social Simulation of a Software Requirements Development Process. Software Process Improvement and Practice, 5, pp. 103-110 (2000).
19. WebDeploy:Workflow – White Paper, Setrag Khoshafian, Technology Deployment International, Inc (1998).
20. Sohlenkamp, M.: Supporting Group Awareness in Multi-User Environments through Perceptualization, M.Sc. dissertation, Fachbereich Mathematik-Informatik der Universität, Denmark (1998)(orgwis.gmd.ed/projects/POLITeam/poliawac/ms-diss/ access on Sep/1999)
21. Janz, B.D.: Self-directed teams in IS: correlates for improved systems development work outcomes. *Information & Management* 35, pp. 171-192 (1999).
22. Perry, D.E., Staudenmayer, N.A., Votta, L.G., People, Organizations, and Process Improvement. IEEE Software, pp. 36-45, July (1994).
23. Ellis, C., Gibbs, S.J. e Rein, G.L., GROUPWARE: some issues and experiences. Communications of the ACM, vol. 34, n.1, pp. 39-58, Jan (1991).
24. Workflow Management Coalition, www.wfmc.org, last access in June/2001.
25. Stenmark, D., Turning Tacit Knowledge Tangible, In: Proceedings of the 33rd Hawaii International Conference on System Sciences, (2000).
26. Proceedings of the ACM Conference on Computer Supported Cooperative Work - CSCW (since 1988).

Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling

Ulrike Becker-Kornstaedt

Fraunhofer IESE, Sauerwiesen 6, D-67661 Kaiserslautern, Germany
becker@iese.fhg.de

Abstract. Capturing a process as it is being executed in a descriptive process model is a key activity in process improvement. Performing descriptive process modeling in industry environments is hindered by factors such as dispersed process knowledge or inconsistent understanding of the process among different project members. A systematic approach can alleviate some of the problems. This paper sketches fundamental difficulties in gaining process knowledge and describes a systematic approach to process elicitation. The approach employs techniques from other domains like social sciences that have been tailored to the process elicitation context and places them in a decision framework that gives guidance on selecting appropriate techniques in specific modeling situations. Initial experience with the approach is reported.

1 Introduction

Descriptive software process modeling attempts to determine the actual processes used in an organization [1]. Descriptive process models are key assets in the context of software process improvement programs. They describe current development practices within an organization at a rather high level of detail, and consequently can help detect weaknesses in the process [2].

A major quality criterion for a descriptive software process model is its *accuracy* (i.e., the degree to which the model reflects the actual process), so that any process-related activities can be based on the process as it is carried out. A Process Engineer who uses an inaccurate process model, for instance as a basis for a measurement program, may try to measure activities that are only depicted in the model but do not take place in reality.

Process elicitation in industry is complicated by several factors, such as a high number of different roles in the process, their limited availability, or dispersed process knowledge. However, little research has been done in developing methods for capturing descriptive process information [1]. A first step towards developing such an approach is to adapt techniques developed for similar areas to the software process elicitation context. To guide a Process Engineer in the usage of these techniques, a decision framework is required. This paper presents the development of an approach to systematic process elicitation and first validation results.

The paper is structured as follows: Section 2 gives an introduction to descriptive process modeling and describes the difficulties a Process Engineer encounters in

capturing real-world processes. Section 3 proposes the structure of a systematic approach to process knowledge elicitation and introduces its components. Section 4 presents a brief validation of the approach. Section 5 discusses related work. Finally, Section 6 concludes with a summary and outlook.

2 Descriptive Process Modeling

This section sketches the context in which descriptive software process modeling is usually performed in industrial environments.

2.1 Context of Process Modeling

Usually, process modeling is part of a larger process improvement program. The model may be used as a foundation for measurement, to detect weaknesses in the process, or to design a new process based on current practices. Typically, process modeling is performed by a Process Engineer external to the target organization.

Descriptions of the *actual process* [3] i.e., descriptions of how the development process actually takes place in reality, are important assets in software engineering in general and in software process improvement in particular. One key concept for process improvement is to incrementally incorporate experience gained and lessons learned from carrying out the process. Capturing the process as is in an explicit description, in a descriptive software process model, is therefore a first step. Changes to this as-is process can then be incorporated into the model, and the updated and improved process can be disseminated to process participants [2]. The model can be used for a detailed process analysis to detect weaknesses, to define measurement points in the context of measurement programs, to help define metrics and facilitate data collection, or to capture experience. An explicit description of the process is also an important basis for audits, assessments, or certifications. Figure 1 explains the role of descriptive software process models in process improvement and the major steps in descriptive software process modeling, elicitation of process knowledge, creation of the model, and review of the model [3].

The schema presented in [4] names the following information to be described in a process:

- activities, i.e., what is being done,
- artifacts, i.e., what is being used and produced, and
- agents, i.e., descriptions of the responsibilities.

In addition to these, relationships between them, such as product flow, role assignment, or refinement have to be described.

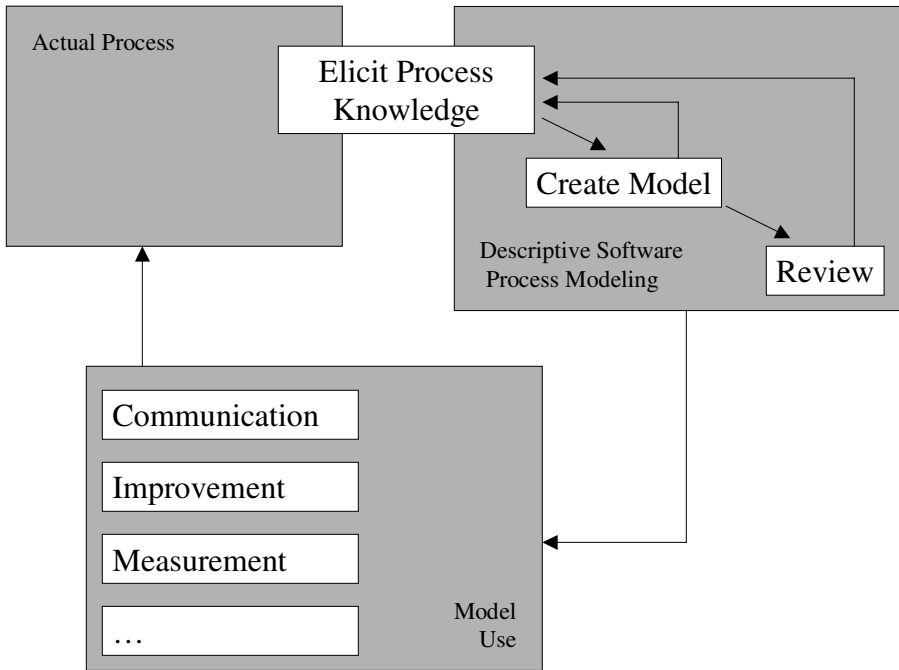


Fig. 1. Steps in descriptive software process modeling

However, what information needs to be mapped onto the model, or the level of detail to be covered depends on the specific purpose of the model, and its intended users.

The major tasks in developing a descriptive model are elicitation of process knowledge, formalization of the process knowledge to create a model, and review of the model [5]. Usually, there are rework cycles going from model creation back to elicitation, for instance, when important information is missing, and from review to the elicitation step, if it shows that the model is incomplete or incorrect. The elicitation step is the first one in process modeling on which the other steps build. Thus, the result of the elicitation step is crucial.

Typical information sources for software process knowledge reported in the literature are analysis of process artifacts, observation of process performers, and interviews with Process Performers (see, for instance, [3]).

Analysis of Process Artifacts. Analysis of artifacts refers to examination of artifacts produced by a group under study. In the case of process elicitation, two categories of artifacts can be distinguished: Process artifacts and process documentation.

Process artifacts are the artifacts developed as intermediate or final products in the process. Process artifacts provide evidence for the actual process. Examples of process artifacts are requirements or code documents. Process artifacts can give information about themselves, their contents, and their structure. Relationships between the artifacts and the activities producing these artifacts are sometimes included.

Process documentation refers to documents that describe how the process is to be performed and what needs to be done to obtain the expected results. Examples for process documentation are process handbooks or project plans. Process documentation has the advantage that it usually describes the relationships between different information entities, such as which documents an activity should produce, or which role is supposed to perform an activity. However, process documentation gives information on the official process [3] (i.e., the process as it should be performed). This information does not necessarily have to match the actual process.

The major strengths of using existing documents are that they can be reviewed repeatedly, and that they are unobtrusive [6].

Observation of Process Performers. Observation refers to the selection and recording of a set of behaviors in their natural ‘environment’. One major problem of observation is that only those events that occur during observation can be captured. In addition, observation is often perceived as obtrusive.

[7] describes how observation of Process Performers was used to find out about the actual process. [8] propose to use data collected on the events of an executed process to discover the behavioral aspects of the process. However, this assumes that people already collect data on the process and know what data to collect.

Interviews with Process Performers. Interviews with Process Performers are flexible with respect to information content and level of detail. However, Process Performers can only report on their view of the process.

2.2 Problems in Process Modeling

Literature sources (for instance [3]) describe possible sources for process information. But they do not provide details on *how* to extract the process knowledge from these sources, what techniques to use to exploit them, or under which conditions to exploit which type of information source. |

One characteristic of software processes is that they are creative, human-based processes. Other than most manufacturing processes, where always the same product is being produced – apart from minor modifications – , software processes deal with individual solutions.

Some of the problems that often make software process elicitation in industrial practice difficult include:

- **P1:** Knowledge about the process is dispersed within the organization due to the complexity of software development. This complexity implies that a high number of Roles is involved, each of them with very specific tasks in the process, and each of them is only involved in a fraction of the overall development process. [9] It is impractical to interview everybody involved in a process. Thus, sampling strategies for interviews are needed.
- **P2:** In large teams, especially different parts of the software process may be performed in different geographical locations. This makes it more difficult to get a consistent and complete picture of the development process.
- **P3:** Depending on their roles in the process, different Process Performers have individual views of the process [9]. Union of the views may lead to an inconsistent global picture. This is especially the case when the process is very complex, and

when the different Process Performers are exclusively involved in their parts of the process.

- **P4:** Some process steps are performed very seldom. Observation of such process steps cannot be planned, and Process Performers tend to forget about these steps or are not able to report them accurately.
- **P5:** Process Performers may leave out aspects of the process [10]. Often, Process Engineers may not be aware of getting incomplete information. Thus, it is very important that Process Engineers have some background information so that they can assess the information provided by Process Performers.
- **P6:** Access to Process Experts is difficult to obtain [11]. Experienced Process Performers are not only the most favored interview partners for process elicitation, but the more experienced Process Performers are, the more they are involved in projects, and the tighter are their schedules, and the more difficult it is to arrange interviews with them.
- **P7:** Process Engineers from outside the target organization do not know the organization well enough to judge who is the right expert to provide the information needed. Badly selected interview partners may give inadequate information (such as process steps other than those asked for, or the Process Performer may not be able to provide the knowledge at the level of granularity requested). On the one hand, this is wasted effort for the people interviewed; on the other hand, this usually leads to additional cycles in modeling and review.
- **P8:** External Process Engineers, being unfamiliar with the company-specific terminology, may misinterpret information and develop an inaccurate process model. The more detailed the process information, the more difficult it is to interpret it in the right context.

The problems mentioned above lead to incorrect or incomplete process models, or models that do not fully suit their intended purpose. Such models cause additional effort from both Process Engineers and Process Performers for re-interviews, modifications, and additional reviews.

Thus, process elicitation needs to be performed in a way that is efficient for Process Performers and for the Process Engineer, meaning it avoids inefficient interviews (for instance, because a Process Performer is interviewed who can not give the information required) and rework (i.e., re-interviews, modification, and additional reviews) because of misunderstanding or incomplete information and additional review cycles.

In practice, process elicitation as currently practiced depends very much on the skills and experience of the Process Engineer performing the modeling task. A systematic approach to process knowledge elicitation is needed to alleviate some of the problems mentioned above, and to make software process elicitation efficient for Process Experts and less dependent on the skills and experience of the Process Engineers. The next sections describe some techniques and how they were developed, as well as a decision model that could make them available in practice.

3 Method Design

3.1 Overall Framework

A large number of the problems reported in Section 2 can be alleviated if a Process Engineer already has some background knowledge on the organization before getting involved in details of the process. Thus, process elicitation should be performed in two stages: process familiarization and detailed elicitation [11], as depicted in Figure 2. Figure 2 details the descriptive process modeling step as described in Figure 1. To each of these stages, appropriate techniques have to be associated.

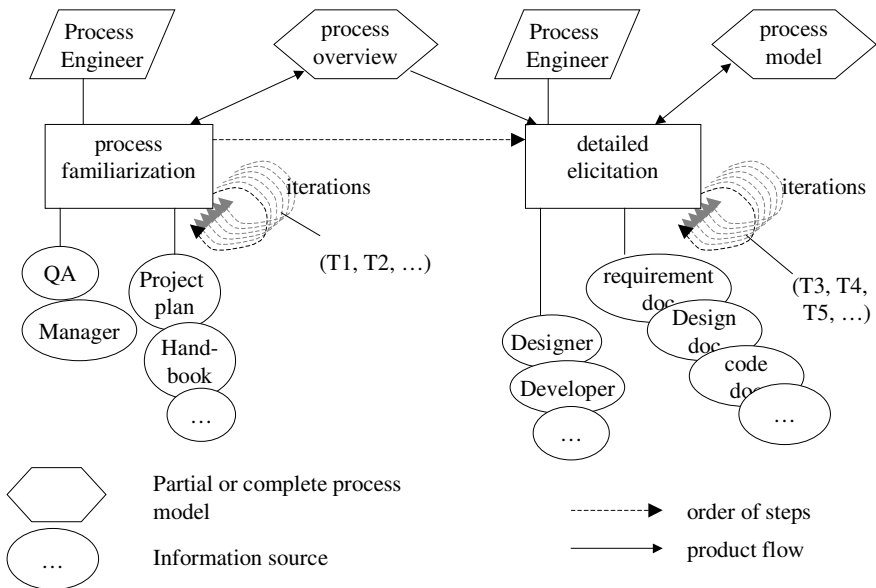


Fig. 2. Process elicitation

The aim of the *process familiarization phase* is to obtain an overview of the process and its general structure. Typical human information sources in this situation are Managers and Quality Assurance people. Documents helpful at this stage include process handbooks or project plans that give an overview of the process. In this stage, techniques that give an overview of the process are mainly used for knowledge elicitation ($T1, T2, \dots$). The techniques have to be characterized with respect to their scope (i.e., what type of knowledge can they elicit) and under which circumstances they are to be used. Having an overview of the process can facilitate elicitation in the later stages and relieve some of the problems described above. For instance, an

overview of the process can help determine which roles need to be asked (P1), or may help the Process Engineer understand where inconsistencies in process understanding result from (P2). Process familiarization can alleviate problems related to not knowing the target organization (P7), (P8). The result of this stage is an overview of the process structure, such as the major activities and artifacts, and possibly the product flow among them.

In the second phase, the *detailed elicitation phase*, the Process Engineer tries to obtain details on the process, such as detailed descriptions of activities or artifacts. During this stage, example artifacts from the process can provide valuable information. Typical human information sources are Designers or Developers, i.e., roles that have a very detailed view of their part in the process. Typical documents to look at are example documents, such as requirements documents, design documents, or code documents. Techniques employed in this stage (T3, T4, T5, ...) are mostly techniques that allow to obtain in-depth information.

3.2 Method Design

To further fill the framework described in the previous section, concrete techniques have to be defined.

Process elicitation is done over many iterations. In each iteration step the process model is further refined, completed, or inconsistencies are removed from the model. In each iteration step, a Process Engineer must decide what information is missing in the current version of the process model. Accordingly, he has to select a technique and source that allow him to elicit missing information to further complete the process model.

As process modeling projects and their contexts differ a lot from one another regarding issues such as their goals, the requirements for the model, or the environment in which process modeling takes place, it can not be expected that a single procedure or a deterministic algorithm describing the order of techniques to apply will be adequate to suit every process modeling project.

Instead, a systematic approach to process elicitation will rather consist of a set of techniques specifically designed or tailored to software process knowledge, and a set of heuristics or guidelines (collectively called a *decision model*) describing when to apply which technique and what the expected outcome is. For instance, a Process Engineer who has no further information on the overall process will most likely not be able to use information on the details of a coding activity, as the overall context of the activity is unclear to him.

The major factors determining whether a technique can be applied depends on what knowledge is already available, what knowledge is still missing, and the information sources available.

Thus, to systematically describe a software process elicitation method, the following components are used:

- A schema describing what the building blocks of the method, i.e., the techniques used in the method have to look like. This schema allows to characterize the interfaces between different building blocks, and it allows to extend the method by developing new building blocks.

- A schema that allows to classify process information to describe process models in terms of completeness as well as knowledge that still needs to be elicited.
- Concrete building blocks to instantiate the framework for the method. These building blocks may be domain-specific.

3.3 Techniques

Knowledge elicitation from artifacts and interviews with, or observation of, Process Performers has been done in other disciplines. Qualitative research methods refer to research procedures that produce descriptive data: people's own written or spoken words and observable behavior [12].

Qualitative research techniques have been used for a long time, especially in social sciences. In addition, these techniques have been tailored to other, more technical, contexts, for instance, Software Engineering [13], or in Expert System design. Thus, experience does not only exist with respect to the usage of these techniques, but also regarding the tailoring and adaptation of the techniques to other contexts.

To develop a systematic approach to software process elicitation, techniques from other disciplines have to be (and in part were) explored for their applicability to software process elicitation.

A field that has tailored qualitative research techniques to solve its own problems in a more technical context is the Knowledge Acquisition (KA), i.e., 'the activity of gathering knowledge for especially knowledge-intensive programs' [14].

Techniques that have already been employed in the KA context are therefore good candidates for tailoring to a method for systematic knowledge elicitation in the context of software process modeling.

Process elicitation techniques are not only needed to elicit knowledge from different sources, but also for different stages of process elicitation, requiring different types of knowledge, and different degrees of detail. These adapted techniques take into account the specifics of software processes and software process knowledge.

3.4 Schema for Building Blocks of the Method

To select the most promising techniques applicable in their current context, taking into account characteristics of the software process and the organization, Process Engineers need heuristics.

The applicability of a technique depends on:

- what stage the technique is intended to cover,
- the type and detail of knowledge that should be elicited in the next step,
- the information sources currently available, and possible techniques to exploit these sources,
- premises for using a technique (for instance, a technique may require a Process Engineer to have background knowledge on the process, as pointed out by P5, or needs to know the organizational structure),
- which role or artifact can provide the knowledge needed (e.g., a Manager or a Tester will not be able to provide detailed information on activities and artifacts

- related to requirements specification). This may require sampling strategies (see 1) (*Source*),
- possible risks of applying a technique. For instance, when using techniques that involve human experts, they report their own view of the process – which may be different from the actual process (see P3) (*Risks*).

Process knowledge is difficult to describe in terms of completeness. Completeness of process knowledge is difficult to judge, as there is no ‘reference body of knowledge’. However, what can be described more precisely is the type of information elicited (i.e., can the technique elicit information regarding activities or artifacts) with respect to its usage in the process model. One way to characterize software process knowledge is to map process knowledge onto existing software process modeling schemas, such as the one described and implemented in the Spearmint tool [15]. This schema describes the major information entities of a process model, i.e., activities, artifacts, roles, as well as the relationships product flow, role assignment, and refinement of these entities. To further describe a process, it is possible to attach user-defined attributes to the process entities.

Within this framework, process knowledge can be further classified according to semantic properties. For instance, process knowledge could be classified as ‘a detailed description of an activity’. Employing such a schema, it is possible to describe in terms of schema elements what information is already captured in the model and what information is missing. In addition, using defined techniques allows tracing the different steps followed in elicitation, and the intermediate results.

An additional benefit of tying the process knowledge to be elicited to a process modeling schema that is implemented in a tool is that the technique allows tracing process knowledge down to the process model. Another extension of the elicitation method is that it can be directly coupled with the tool.

A technique used in the approach could look like the following:

Technique: Focused Discussion based upon Process Model Diagram

Source: Human Process Expert

Stage: Familiarization

Description: Structured interview that focuses on structural properties of the process (e.g., product flow, control flow). During the interview, the Process Engineer develops a graphical representation of the process according to the information provided by the interviewee. The diagram is discussed with the interviewee, and possibly refined further. This technique is applicable to elicit structural properties of the process.

Derived from: Focused Discussion. A focused discussion is a structured interview where the topic of the interview centers on a specific type of information such as situations, artifacts, or diagrams [5], [16]. A focused discussion related to process artifacts can help elicit details regarding the artifact, who was involved in producing the artifact, what are the steps, who uses the artifact as input, etc.

Knowledge required to start: none; this technique is suitable for early stages of process elicitation, to obtain an overview of the process structure.

Type of knowledge elicited: activities, artifacts, product flow, control flow between activities

Granularity: overview of process

Additional material: none

Remark: This technique is especially suited to give an overview of the process structure in the early stages of elicitation. The structure of the process model is a very good starting point to elicit more details on the different process entities.

Risks:

- Process Engineer uses a notation that is not familiar to Process Performer.
- Since the Process Engineer has no background knowledge the Process Performer can dominate the interview.
- Process Performers confound the official process (as may be described in a handbook) with the actual one (the one taking place in reality).
- Interviewees may report the official process, because they think this is what the interviewer wants to hear .

4 Validation of Methodology

When validating a process modeling approach, one has to be aware that the quality of the resulting model and the efficiency of the modeling process may be influenced by factors other than the method itself. One of these factors is for instance the experience of the Process Engineer: An experienced Process Engineer without an explicit method may achieve a model of higher quality than an inexperienced Process Engineer with a sophisticated method.

Thus, the best way to evaluate knowledge elicitation technique is to use case studies. Although case studies cannot prove the applicability of a technique under all possible circumstances, they are a good means to show the capability and deficiencies of a technique in one situation.

4.1 Experience with Approach

This section presents a first case study where process elicitation was done following the approach described above. The process model was to describe the major activities, artifacts, and roles in the development process, as well as product flow and role assignment.

In this case, little background material on the process was available in the beginning. For process familiarization it was therefore essential to first obtain an overview of the structure of the process, i.e., the relevant activities, artifacts, and the product flow between them. Thus, it was decided to use a focused discussion, based on a diagram to obtain background knowledge of the process.

We used this technique in a group interview involving five developers. In the interview, we first asked for the relevant activities, then for each of the activities, the artifacts related to them. For activities and artifacts we had used a very simple notation: paper notes of different shapes. These were put up on a whiteboard. Product flow could be easily drawn between those process entities. Using a whiteboard facilitated discussion and modification of the process model. One problem was that the picture soon became too large and complex, resulting in an unclear, cluttered

picture. The technique proved to be very well suited to obtain a good overview of the process structure. In addition, the focused discussion showed to be very useful as a group technique. The Process Engineer was rather the moderator of the group than an enquiring interviewer, as the group discussion had synergy effects. One major advantage of the group discussion was that inconsistencies in the Process Performers' understanding of the process (e.g., developers' opinions differed with respect to whether a certain artifact was needed in an activity) could be discussed and resolved directly. The information gained from the group discussion reflects more than one person's view. The group interview also increased the Process Performers' understanding of the process.

[12] describe that a major risk of all group interview techniques is that people cannot be expected to say the same things in a group that they might say in private. When using this technique for Process Elicitation, Process Performers may not report the actual process for fear of consequences when their superiors are present. In our case, this did not happen. Individual interviews that were conducted afterwards did not reveal differences to the process model elicited in the group interview.

One problem we encountered was related to the difference between product flow and control flow that was often confused. Whereas product flow only denotes which artifacts are used and produced in which activities, this does not necessarily determine the exact order, the control flow, of activities. This experience will be incorporated into the next version of the decision model as a risk.

The process knowledge gained using this technique showed to be a good foundation for further detailed interviews. Furthermore, due to the good overview, sampling of interviewees could be done fairly easy – based on their roles in the process (see P1). These roles had been identified in the first stage of process elicitation.

During the detailed elicitation stage, the process model was further refined. It showed especially, that there were very few inconsistencies that had to be resolved, as the overall structure of the model was clear.

5 Related Work

This section describes frameworks that have been developed for the elicitation of software process knowledge. [17] describes the *Elicit* approach, a general strategy to develop descriptive software process models. The approach gives a good general overview of the core steps of elicitation - comprising activities such as *understand the general organization*, *plan the elicitation strategy*, and *elicit the process knowledge* needed. The elicitation step is based on document analysis: reading and identifying the structure of selected process documents. However, Elicit does not take into account that the majority of process information is typically provided by Process Performers. In addition, more detailed techniques are missing in the Elicit approach.

[9] introduces an approach called *Multi-View-Modeling*, consisting of three steps. In the first step, different role-specific process models, so-called *views*, are developed. In the second step, the different views are checked for similarity and consistency. After resolving potential inconsistencies, the views are integrated to form a comprehensive software process model in a third step. However, the approach does not give hints on

how to systematically obtain the knowledge needed to create the different views of the process model from the various process experts.

[18] describes a strategy to develop (business) process models in his *process modeling cookbook*. The method suggests that a process modeler should gather as much initial information from conversations with process participants and should leave aside any official documents and verbal support. The process modeling cookbook claims that official process documentation may be misleading. For process elicitation, the author suggests using interviews. However, the techniques described in the cookbook focus on the creation of the model, and not on process elicitation.

[8] suggest capturing data from events that take place during the software process, whereas an event is anything that can be observed in the course of the process. However, the approach does not describe how to systematically evaluate those events to come up with a description of the process.

6 Summary and Outlook

This paper describes the situation in descriptive software process modeling and the need for a more systematic approach to software process elicitation. The design of a more systematic approach is sketched. This approach consists of a set of techniques and a set of selection criteria for each of the techniques. The techniques are adapted from other fields that have already used knowledge elicitation in similar contexts. The selection criteria describe the context under which a technique is likely to be successful, what type of knowledge it is able to elicit, what type of background knowledge is needed to apply the technique, and what are possible risks when applying the technique. The selection criteria take the specific problems of software process elicitation into account.

This approach is a first framework towards more systematic process elicitation. Although software process knowledge cannot be characterized with respect to its completeness, since a reference body of knowledge is not available, knowledge can be characterized with respect to its type, and – to some degree – its level of detail. One means for this characterization is to use an existing process modeling schema. Other factors that have an impact on which technique to apply are formulated as entry criteria and risks.

As software process elicitation deals in large parts with humans, process elicitation will always need a high degree of flexibility. Using this approach allows analysis of the approach and learning. The structure presented here is a first step towards more systematic process elicitation. However, an important aspect will be to incrementally extend the approach. Criteria may have to be revised or refined, as more experience in the application of the techniques is available.

Acknowledgements

The author would like to thank Leif Kornstaedt and Jürgen Münch for the inspiring and fruitful discussions on descriptive software process modeling and systematic

process elicitation. Additionally, she would like to thank Sonnhild Namingha for her *systematic* proofreading of the paper.

References

- [1] Bill Curtis, Marc I. Kellner, and Jim Over. Process Modeling. Communications of the ACM, 35 (9): 75-90, September 1992. K. E. Huff. Software process modelling. In A. Fuggetta and A. Wolf, editors, *Software Process, Trends in Software*, chapter 1, pages 1-24. John Wiley & Sons, 1996.
- [2] K. E. Huff. Software process modelling. In A. Fuggetta and A. Wolf, editors, *Software Process, Trends in Software*, chapter 1, pages 1-24. John Wiley & Sons, 1996.
- [3] S. Bandinelli, A. Fuggetta, L. Lavazza, M. Loi, and G. P. Picco. Modeling and improving an industrial software process. *IEEE Transactions on Software Engineering*, 21(5):440-454, May 1995.
- [4] J. W. Armitage and M. I. Kellner. A conceptual schema for process definitions and models. In D. E. Perry, editor, *Proceedings of the Third International Conference on the Software Process*, pages 153-165. IEEE Computer Society Press, October 1994.
- [5] E. S. Cordingly. *Knowledge Elicitation Principles, Techniques, and Applications*, chapter Knowledge Elicitation Techniques for Knowledge-based Systems, pages 89-172. Ellis Horwood Limited, Chichester, Great Britain, 1989.
- [6] R. K. Yin. *Case Study Research: Design and Methods*. Sage, 2nd edition, 1994.
- [7] D. B. Walz, J. J. Elam, and B. Curtis. Inside a software design team: knowledge acquisition, sharing, and integration. *Communications of the ACM*, 36(10):63-77, October 1993.
- [8] J. Cook and A. Wolf. Automating process discovery through event-data analysis. In *Proceedings of the Seventeenth International Conference on Software Engineering*, pages 73 - 82. Association of Computing Machinery, April 1995.
- [9] M. Verlage. An approach for capturing large software development processes by integration of views modeled independently. In *Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering*, pages 227-235, San Francisco Bay, CA, USA, June 1998. Knowledge Systems Institute, Skokie, Illinois, USA.
- [10] M. I. Kellner and G. A. Hansen. Software process modeling: A case study. In *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, volume II, pages 175-188, 1989.
- [11] U. Becker-Kornstaedt and W. Belau. Descriptive Process Modeling in an Industrial Environment: Experience and Guidelines. In R. Conradi, editor, *Proceedings of the 7th European Workshop on Software Process Technology (EWSPT 7)*, Kaprun, Austria, pages 177-189, Lecture Notes in Computer Sciences, Springer-Verlag, 2000.
- [12] S. J. Taylor and R. Bogdan. *Introduction to Qualitative Research Methods: A Guidebook and Resource, Third Edition*. John Wiley and Sons, 3 edition, 1998.
- [13] C. B. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering*, 1999.
- [14] H. Eriksson. A survey of knowledge acquisition techniques and tools and their relationship to software engineering. *Journal of Systems and Software*, (19):97-107, 1992.
- [15] U. Becker-Kornstaedt, D. Hamann, R. Kempkens, P. Rösch, M. Verlage, R. Webby, and J. Zettel. Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance. In *Proceedings of the Eleventh Conference on Advanced Information Systems Engineering (CAISE '99)*, Heidelberg, Germany, June 1999. Lecture Notes in Computer Science, Springer-Verlag.

- [16] N. J. Cooke. Varieties of Knowledge Elicitation Techniques. *International Journal of Human-Computer Studies*, 41:801–849, 1994.
- [17] N. H. Madhavji, D. Hölzje, W. Hong, and T. Bruckhaus. Elicit: A Method for Eliciting Process Models. In D. E. Perry, editor, *Proceedings of the Third International Conference on the Software Process*, pages 111–122. IEEE Computer Society Press, October 1994.
- [18] P. Kawalek. The Process Modelling Cookbook: Orientation, Description, Experience. In *Proceedings of the 2nd European Workshop on Software Process Technology (EWSPT 2)*, pages 227, 1992.

Modular Process Patterns Supporting an Evolutionary Software Development Process¹

Michael Gnatz, Frank Marschall, Gerhard Popp,
Andreas Rausch, and Wolfgang Schwerin

Technische Universität München, Institut für Informatik,
Lehrstuhl Professor Dr. Manfred Broy,
D-80290 München, Germany
{gnatzm, marschal, popp, rausch, schwerin}@in.tum.de

Abstract. Change and evolution of business and technology imply change and evolution of development processes. Besides that for a certain enterprise and/or project we will usually integrate elements from a variety of existing process models, comprising generic standards as well as specific development methods. In this paper we propose a Process Framework which is modularly structured on the basis of the concept of process patterns. This framework allows us to describe development processes in a way such that change, evolution, and integration of processes are facilitated. Founded on our framework we sketch the idea of a living development process. An example illustrates our approach.

1 Introduction

Nowadays, many different process models exist. These models range from generic ones, like the waterfall model [21] or the spiral model [6], to detailed models defining not only major activities and their order of execution but also proposing specific notations and techniques of application. Examples of the latter kind are the Objectory Process [13], the Unified Software Development Process [15], the Catalysis Approach [11], the V-Modell 97 [12], or eXtrem Programming [4] – just to name some of them.

All these process models have their individual assets and drawbacks. Hence, one would wish to take all the different assets and benefits of the various process models as a basic construction kit for an integrated development process tailored to the specific needs of the individual team, project, company, and customer. Jacobson, for example, talks about the unified process as a “strawman” serving only for explanatory purposes and probably never applied exactly as proposed [14].

¹ This work originates from the research project *ZEN – Center for Technology, Methodology and Management of Software & Systems Development* – a part of *Bayerischer Forschungsvverbund Software-Engineering (FORSOFT)*, supported by the *Bayerische Forschungstiftung*.

To assemble a specific development process from existing models we have to identify and characterize explicitly the building blocks and their relations of a process model in general. Therefore we need a set of basic notions and definitions common for all process models – the Process Framework. This Process Framework must allow us to integrate the various existing process models. The Process Framework can serve as a common basis for the definition of a development process that incorporates the assets and benefits of the different existing process models and that can be flexibly adapted to different kinds of project requirements and situations.

Once you have defined your standardized development process in terms of the Process Framework, you still have to adapt this development process to different projects and project situations. This is often referred to as static tailoring. But, our business is changing almost every day: the requirements of our customers change, new technology has to be adopted, and finally the way we work together evolves. To be successful in a changing environment we not only need static adaptation but also a more flexible way of adaptation - the dynamic adaptation.

Tom DeMarco even mentioned about the nature of process models and methodologies in [10]: „It doesn't reside in a fat book, but rather inside the heads of people carrying out the work.“ Thus, our Process Framework must additionally offer the ability to incorporate the process knowledge of the whole company. It must provide a platform for a learning organization recording the evolution steps of a living software development process.

Therefore the Process Framework must be open for the integration of new process elements, for the extension and adaptation to new technologies and fields of application. Besides static adaptation – support of different kinds of projects and project situations – there is a need of dynamic adaptation.

In this paper we propose a Process Framework that is sufficiently powerful to fulfill these requirements. First, in Section 2 we give an overview over different modeling levels of development processes. We present the requirements on and user views of an evolutionary process model. In the next section, Section 3, we define our Process Framework. In Section 4 we present the concept of process patterns, providing guidelines about the organization of the actual development process. A conclusion is given at the end of the paper in Section 5.

2 Basic Concepts of the Living Software Development Process

Various people and groups get into touch with process models and metamodels. In the next section, we discuss the different levels of a software development process. Then, in the following two sections, we present the two main views on process models - the project view and the method view. We will discuss their specific way of interaction with the living software development process we are going to propose in this work. Thus, we can show the needs and benefits of the two different user groups mentioned above.

2.1 Process Models and Metamodels

Developing and maintaining software has become one of the most challenging tasks a company can do. Following some kind of process model, such as the Rational Unified Process [17] or the V-Modell 97 [12], promises to provide guidance facilitating the development task. Usually there are different people involved with different views on the development process itself.

Developers and project leaders are concerned about the development process of their individual projects. They concentrate on concrete tasks and results of their actual project, like the development of an analysis model of the system under consideration. Accordingly to [28, 29] we can divide the software development process into the *production process*, that is the process in charge of developing and maintaining the products to be delivered, and the *meta process*, that is the process in charge of maintaining and evolving the whole software process. Using this terminology, we see the focus of developers on the production process.

Another group of people being primarily concerned with the meta process – especially in large organizations – might be a methodology group. Companies, which are on Capability Maturity Model (CMM) level 3 or higher, have a standardized process model [20]. This standard process model provides guidelines and support for organization's projects in general.

If a company is on CMM level 5, continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies [20]. The organization as a whole and the projects themselves must address continuous realization of measurable software process improvement, like for instance defect prevention, technology change management, and process change management. Thus, the companies' methodology group must be able to improve and evolve the standard software process. Therefore this group needs a common Process Framework capturing the basic concepts of software development process models. Figure 1 illustrates three levels of an overall model for software development processes where the aforementioned views can be mapped on.

The Instance Level in Figure 1 captures those elements that belong to a certain project, such as an analysis document of a concrete project.

The Model Level describes a certain software development process. This process definition contains an outline of an analysis document or a description and guidelines of how to organize and hold a workshop with customers to elicit the requirements. This level offers the guideline and information for project managers as well as team members. A specific *Process Model*, as defined in [28], expressed in a suitable process modeling language, would be an element of our Model Level.

The Metamodel Level provides the basic framework to establish a living process model. It offers clear definitions for terms like „Work Product” or „Activity”. The Metamodel Level represents the common conceptual base of a company's methodology group to improve and evolve the underlying standard software development

process². It is on this level where (the concepts of) process modeling languages, such as EPOS SPELL and SOCCA (cf. [9, 28]) are defined.

Metamodel Level

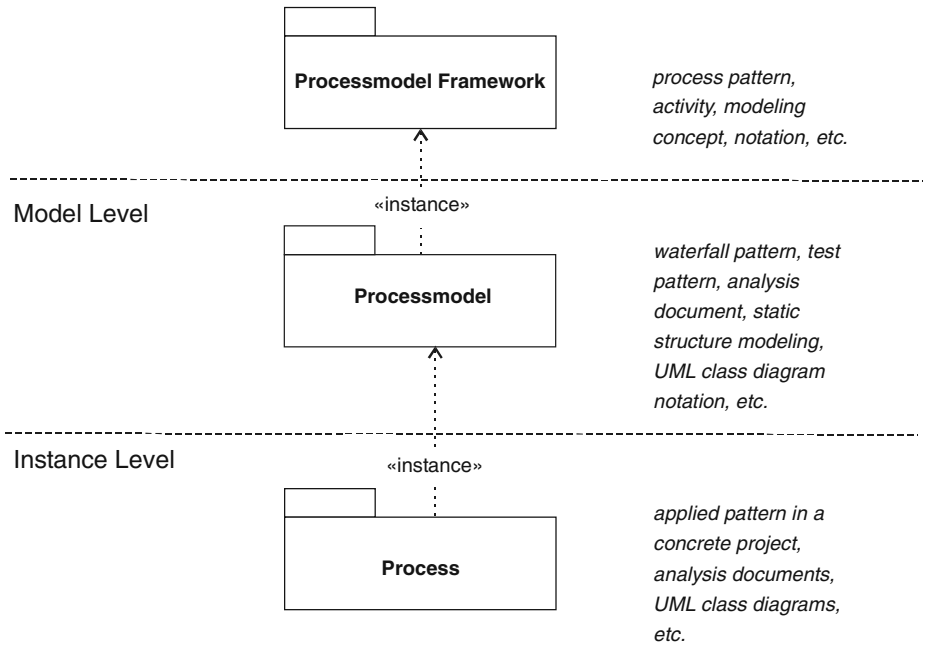


Fig. 1. The Layers of an Overall Process Model

2.2 The Project View of the Living Software Development Process

Project managers and project team members follow the concepts, guidelines, and help provided by a specific software development process defined on the Model Level in Figure 1. While performing their daily tasks they are creating instances on the Instance Level in Figure 1.

Managing a concrete project implies selection of a suitable process from a set of existing, possibly standardized alternatives. Then the chosen process has to be tailored accordingly to the project’s characteristics. This tailored process represents the guidelines, which are to be followed in the project. In terms of our Process Framework, given in section 3, the tailored process defines which work products are to be

² Note, this metamodel structure follows the guidelines provided by the Meta Object Facility (MOF) specification of the Object Management Group (OMG) [19].

produced, and which modeling concepts, notations, and patterns may be applied in the project.

2.3 The Method View of the Living Software Development Process

Process improvement, as required on CMM level 5 [20] for example, means the evolution of process models, i.e. of elements on the Model Level in Figure 1. The formulation of process models on the basis of a well-defined ontology facilitates comprehension and hence changes of development processes. Elements of the Metamodel Level in Figure 1 are supposed to play the role of such an ontology defining terms like “Activity”, “Process Pattern”, “Work Product” and their inter-relations.

An ontology for development processes provides both, developers and the methodology group, with a common vocabulary. On the one hand a methodology group can use such an ontology for the definition of standardized processes. On the other hand developers can use this vocabulary for the description of proposals for changes or additional process elements, which reflect their experience made with former process elements. On the basis of these proposals redefinitions by the methodology group can be done. Figure 2 shows this method view on a living software development process.

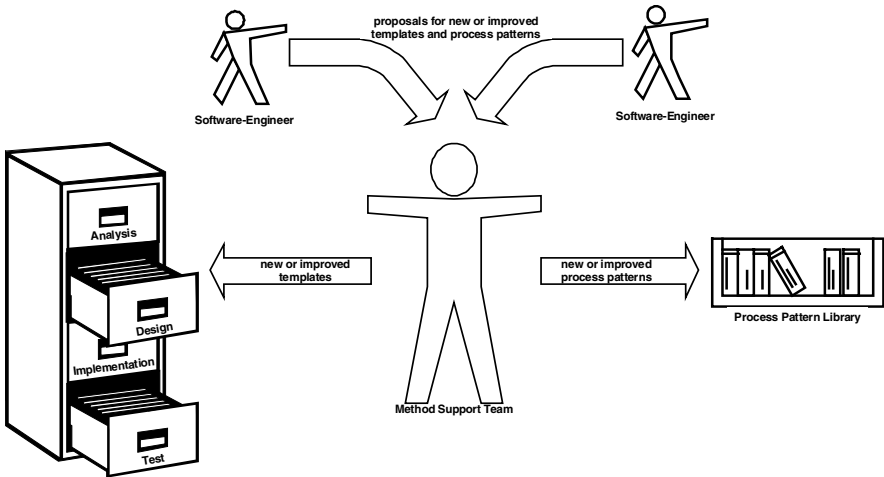


Fig. 2. The Living Software Development Process from the Method View

In our ontology, which we call the Process Framework, given in section 3, we follow the principle of separation of concerns so that changes are facilitated because of having minimal and localized effects.

3 Framework of a Living Software Development Process

In the previous section we have shown how developers and methodology group may interact for elaborating and improving a (standard) software development process establishing a living software development process. Our basic ontology is defined in the Process Framework, which is on the Metamodel Level in Figure 1.

The Process Framework must provide the ability to define and maintain a process model, which integrates elements of all the various existing process models, like for instance the Rational Unified Process [17] or the V-Modell 97 [12]. Thus, the framework must enable the methodology group to state clearly the correlations between the elements of the different process models. Additionally, the Process Framework must support static as well as dynamic adaptation of the process model with respect to the evolution and learning of a living organization (c.f. Section 1).

The new, upcoming concept of process patterns seems to be an approach which basically follows our ideas and which may fulfill our requirements. Process patterns are a very general approach allowing us to integrate existing process models without having to develop a new models from scratch [2, 3, 7, 8]. For example in [1] we have already shown the integration of the V-Modell in the process pattern approach.

The basic idea of the concept of process patterns is to describe and document process knowledge in a structured, well defined, and modular way. Moreover patterns provide information helping us in finding and selecting alternative development steps, similar to strategies and selection guidelines in [22]. Conform with most authors, patterns in our approach consist mainly of an initial context, a result context, a problem description and a solution. The initial context is an overall situation giving rise to a certain recurring problem that may be solved by a general and proven solution. The solution leads to the result context [5].

Figure 3 illustrates the basic concepts of the proposed Process Framework. It develops further the process pattern approach from [7, 8], and integrates it with an enhanced variant of the widely accepted process model framework given in [9]. The framework is based on a clear separation of concerns between the overall result structure, the consistency criteria, and the process patterns themselves.

A Process Pattern defines a general solution to a certain recurring problem. The problem mentions a concrete situation that may arise during the system development. It mentions internal and external forces, namely influences of customers, competitors, component vendors, time and money constraints and requirements. A process pattern suggests an execution of a possibly temporally ordered set of activities. Activities themselves may be carried out following the guidelines of other, subordinated process patterns realizing the activity in question. Therefore process patterns and activities in our framework may be structured hierarchically.

Process Patterns in our framework represent strategies to solve certain problems. Activities represent development steps and are executed by process patterns. An activity does only describe what is to be done but not how it is to be done. In contrast to that a process pattern provides a solution for realizing an activity. Hence generally one activity might be realized by different process patterns. Activities are performed by definite roles. In turn roles are assigned to corresponding persons.

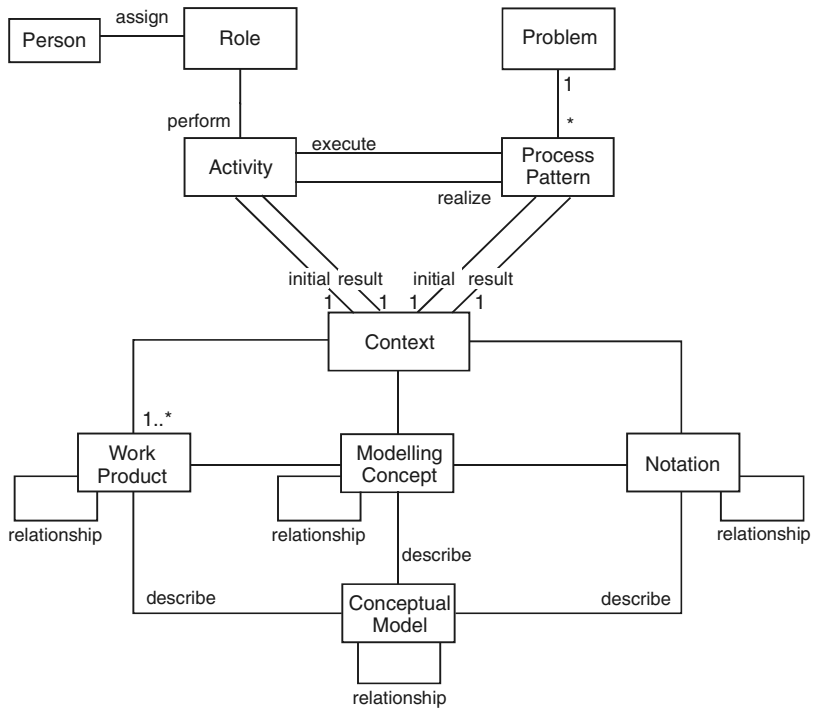


Fig. 3. The Process Framework

Each process pattern as well as each activity needs an initial context to produce a result context. The initial context describes the required project situation to perform an activity or pattern, respectively. The result context describes the situation we reach when performing an activity or pattern, respectively. The context captures the internal state of the development project and can be characterized by constraints over the set of work products. Simple constraints are that certain work products have to exist.

A process model assigns certain process patterns, as for instance the pattern “Planning the Project”, to certain work products, as for example the “Project Schedule”. These work products are described by means of modeling concepts, as for instance “Time Flow Modeling”. The modeling concepts are represented by certain notations, such as “UML Sequence Diagrams”.

The initial and result context of a process pattern may not only require the existence of certain work products, but also that certain modeling concepts and notations are to be applied for these work products. This is important when a pattern proposes the application of notation specific techniques. For instance in [18] methodical guidelines for the refinement of specifications are introduced. These refinement techniques require the modeling concept “Interaction Modeling” based on the notation “Message Sequence Charts”.

For the executes and realizes relationships in Figure 3 we require certain relationships between the contexts of related process patterns and activities. The work prod-

ucts in the result context of a Process Pattern have to be a superset of the result context of each realized activity. The initial context of a process pattern yet has to be a subset of the initial context of each realized activity. With these consistency criteria we cover the intuition that a realizing pattern does require at most the input of the realized activity, to produce at least those results “promised” by the activity.

An example of a realizes-relationship is shown in the context of a Business Process Modeling example in Figure 4. The activity Identify Business Tasks might be realized by the process patterns Inspecting Legacy Systems or Involving Business Experts respectively. The process pattern Inspecting Legacy Systems is a realization of the activities Identify Business Tasks as well as Refine Task Characteristics.

Consistency is also required for the contexts regarding the executes relationship. The union of the result contexts of the executed activities form the result context of the executing process pattern. The initial contexts of the activities have to be subsets of the of the initial contexts of the process pattern they are executed by. Thus intermediate results produced in the workflow of the executed activities need not necessarily be part of the initial context of the executing activity. For an example of the executes-relationship in the context of Business Process Modeling we refer to Figure 4.

The precise definition of the meaning of, and context conditions between work products can be achieved by the use of a so-called conceptual model. Work products that are based on sound description techniques have not only a well-defined notation, but also a possibly even formal semantics in form of a mapping from the set of work products into the set of systems (cf. [16, 23, 25]). A conceptual model characterizes, for instance, the set of all systems that might ever exist. This integrated semantics provides the basis for the specification of a semantic preserving translation from specification work products to program code. This can serve as a basis for correct and comprehensive code generation.

The circular relationship associations assigned to various elements in Figure 3, such as work product and conceptual model, cover the general idea of structuring these elements, for example hierarchically.

4 Process Patterns

A process pattern enables us to describe and document process knowledge in a structured and well-defined way. As already discussed in Section 3 and illustrated in Figure 3 process patterns consist mainly of a problem solved by a proven solution applied through certain activities, and an initial as well as a result context. The activities play important roles in process patterns, because they reflect the tasks, which have to be carried out as described by the solution.

Process patterns, as well as all kinds of patterns, must be presented in an appropriate form to make them easy to understand and discussable. In this section, we first present a uniform pattern description template for process patterns. Then we provide a sample process pattern to illustrate the basic concepts of process patterns.

4.1 Pattern Description Template

A good description helps us grasp the essence of a pattern immediately – what is the problem the pattern addresses, and what is the proposed solution. A good description also provides us with all the details necessary to apply the pattern and to consider the consequences of its application. Moreover a uniform, standardized description template for process patterns is needed. This helps us to compare patterns, especially when we have to select from alternative solutions. For activities similar templates are useful. For brevity we omit a detailed description of this kind of template. Table 1 and Table 2 illustrate our proposal for an activity and process pattern description template by example. A detailed discussion of a wider range of process patterns is not in the scope of this paper and can be found in [7, 8]. Please note, that the sample Process Pattern of this section resides on the Instance Level of the Overall Process Model shown in Figure 1.

The example pattern template given in Table 2 contains, besides others, the following fields:

- **Intent:** A concise summary of the pattern's rationale and intent. It mentions the particular development issue or problem that is addressed by the pattern.
- **Problem:** The problem the pattern addresses, including a discussion of the specific development task, i.e. the realized activity, and its associated forces. Moreover the problem description may contain information with respect to consumers, competitors, and the market situation.
- **Solution:** A solution may suggest certain activities to be applied to solve a certain problem. Possibly an order may be given in which to perform these activities, or alternatives may be proposed. Besides that the solution comprises methodical guidelines and concrete recommendations. A solution shows a possible answer to balance the various forces that drove the project into the current situation. The solution includes a list of activities for execution. In contrast to these activities, the activity realized by the process pattern is referenced below. Moreover the solution depicts the relationships of the initial and result contexts of the executed activities and shows how the activities are combined.
- **Realized Activities:** The name of the activities for which the pattern provides a strategy of how to execute it. Every process pattern realizes at least one activity.
- **Initial Context** The internal state of the development project, i.e. the state of the corresponding work products, that allows the application of this process pattern.
- **Result Context:** The expected situation after the process pattern has been applied, i.e. the resulting state of the work products.
- **Pros and Cons:** A short discussion of the results, consequences, and tradeoffs associated with the pattern. It supports an evaluation of the pattern's usefulness in a concrete project situation. The problem description together with the pros and cons of a pattern helps us in choosing from alternatives, that is static and dynamic tailoring. Thereby these two pattern elements have a purpose similar to selection guidelines in [22].

- **Example:** Known uses of the pattern in practical development projects. These application examples illustrate the acceptance and usefulness of the pattern, but also mention counter-examples and failures.
- **Related Patterns:** A list of related patterns that are either alternatives or useful in conjunction with the described pattern.

In our example we chose the activity Business Process Modeling and a process pattern called Business Process Modeling Task Analysis with Activity Diagrams. This process pattern provides project team members with a strategy for developing a Business Process Model from an Initial Customer Specification.

Table 1 gives the description of the Business Process Modeling Activity:

Table 1. Activity Description: Business Process Modeling (BPM)

Entry	Activity Description
Name	Business Process Modeling (BPM)
Role	Business Expert, Software Architect
Development Issue	The goal of performing this activity is to develop a (complete) business process model, which covers all the business scenarios and business entities described by example in the Initial Customer Specification serving as input. Thereby the project goals, system vision, and constraints specified in the initial customer specification are to be taken into account.
Initial Context	Initial Customer Specification
Result Context	Initial Customer Specification, Business Process Model

Table 2 gives the description of the pattern BPM Task Analysis with Activity Diagrams, which realizes the Business Process Modeling activity:

Table 2. Process Pattern Description: BPM Task Analysis with Activity Diagrams

Entry	Process Pattern Description
Name	BPM Task Analysis with Activity Diagrams
Keywords	Business Process Modeling, UML Activity Diagrams, Stepwise Refinement, Iterated Modeling with Reviews
Intent	Development of <ul style="list-style-type: none">– a precise and unambiguous documentation of a BPM– documentation of BPM on different levels of abstraction cover not only all details but also provide an overview of relevant business processes.– documentation of BPM such that it can be understood by business experts as well as software developers– ensured adequacy of BPM (validated model)
Problem	Business experts are available but a precise documentation of as-is and to-be business processes being relevant for the system to be developed does not exist.

	<p>High complexity of business processes.</p> <p>The system vision of the initial customer specification hints at a strong relationship between the system to be developed and the business processes (e.g. support of large parts of business processes by the intended software system).</p>
Solution	<p>In order to achieve a precise and unambiguous documentation of business processes use UML activity diagrams [26] with its formal syntax to describe business processes.</p> <p>In order to achieve a documentation of different layers of abstraction apply the principle of stepwise refinement. Start with the definition of major tasks and refine them iteratively.</p> <p>Ensure adequacy of the business process model by reviewing each iteration of the model with (third party) experts.</p> <p>Involve business and software architecture experts being fluent with activity diagrams.</p> <p>The patterns workflow - namely its executed activities - is the following³</p> <p>After having identified major tasks and user classes assign user classes as actors to tasks. Refine task characteristics of major tasks, and define a first version of the tasks' task chains by decomposing it into sub-tasks, and defining their causal dependencies. Consider alternative chains.</p> <p>Review this first model involving persons representing the identified user classes in the review.</p> <p>Perform the refinement steps of tasks iteratively and review each iteration (apply pattern Refinement of Activity Diagrams).</p>
Realized Activities	Business Process Modeling
Initial Context	Initial Customer Specification with arbitrary modeling concepts and notations
Result Context	Business Process Model with UML activity diagrams as notation for task chains and a pre- and post-condition style specification of tasks.
Pros and Cons	<p>Pros:</p> <ul style="list-style-type: none">– UML Activity Diagrams provide a standardized, concise and unambiguous notation to document business processes (Task Chains). The applied modeling concepts are widely used by business experts (e.g. similarity with Event Driven Process Chains) [24] as well as software developers (e.g. similarity with Petri nets).– This precise way of description supports detailed and precise review.

³ The temporal ordering of the activities, which are executed by a Process Pattern, may be described by an UML Activity Diagram.

- Iterative modeling and review increases understanding and quality of the business model.

Cons:

- Usage of specific notations, namely UML activity diagrams, may require training of involved persons. A common understanding of the notation must be ensured.
- Stepwise refinement is a pure top down approach so that consideration of existing parts may be difficult.

Related Patterns See also: BPM Informal Task Analysis

Besides BPM Task Analysis with Activity Diagrams, as mentioned in Table 2, the further ways for performing activity Business Process Modeling might exist. For example the process pattern BPM Informal Task Analysis represents an alternative strategy for business modeling proposing an informal documentation of business processes. This might be suitable when business processes are simple and the software system does not play a major role in these processes. The pattern map given in Figure 4 shows these two alternative performance strategies for the business modeling activity.

4.2 Managing the Process Lifecycle – Process Pattern Maps

As already mentioned process patterns will exist on several levels of detail. Like in other pattern-based approaches, a single pattern may be combined with others, forming the lifecycle of a development process. This lifecycle will not be fixed, but will vary from project to project and from situation to situation, according to the specific problems of the projects.

The selection of a process pattern may be outlined as follows: Based on the project's current situation, as partly represented by the state and consistency of work products, the project leader tries to identify the next activities he wants to be executed. This information leads to the selection of one or more alternative process patterns with initial contexts and problem descriptions matching the current situation. After a careful consideration of the alternatives' pros and cons and their problem descriptions one pattern is chosen. This pattern recommends a number of development activities and their temporal order. For each of its activities the solution may require or propose the application of certain process patterns.

By choosing process patterns the project manager forms the process lifecycle. Usually a process pattern library will contain a large number of patterns. We introduce two kinds of pattern maps providing an overview over a set of patterns by structuring them from different points of view.

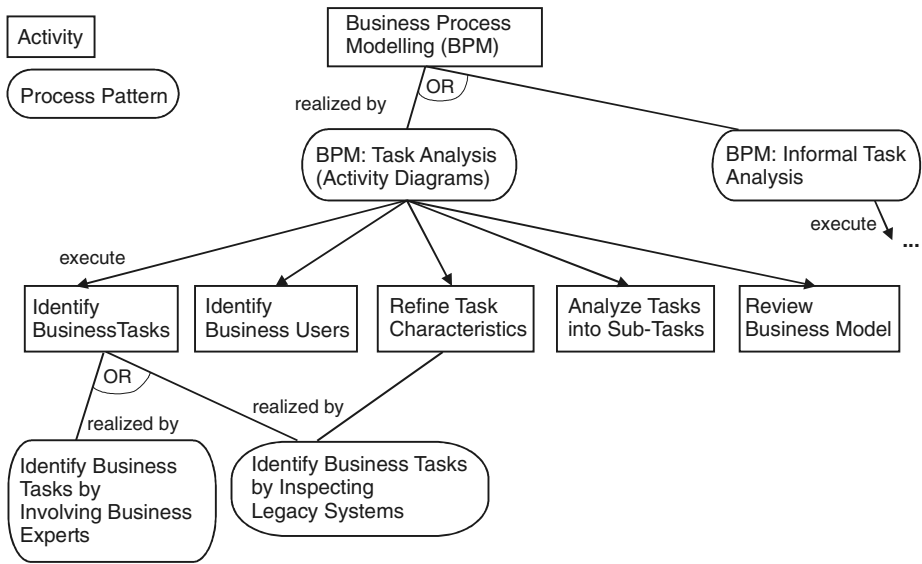


Fig. 4. An Activity Process Pattern Map

One possibility is to structure process patterns accordingly to the activities they realize. We call this activity process pattern map. Activity process pattern maps are directed graphs. These graphs have two kinds of nodes, namely activities and process patterns. A process pattern node has edges to all the activities that have to be performed by applying the pattern and exactly one edge to the activity it realizes. Each activity may be performed following the solution provided by a process pattern. Hence each activity node has edges to process pattern nodes that provide guidelines to perform the activity. Figure 4 illustrates such an activity process pattern map that builds a graph.

A second viewpoint on a set of pattern maps is the so-called context process pattern map. This kind of map is a directed graph with contexts as nodes and patterns as arcs. This way we can easily see alternative ways, i.e. process patterns, from one context to another. These maps are similar to the maps presented in [22].

5 Conclusion and Further Work

In this paper we enhanced existing Process Frameworks comprising elements, such as work product, activity, and role, by introducing the notion of process pattern as a modular way for the documentation of development knowledge. According to the best of breed idea, a company can combine the most appropriate solutions from different established methods and best practices to form a perfectly tailored process for a company and for all types of projects. Therefore we introduced a Process Framework that facilitates integration, tailoring, and evolution of process models.

Similar to the approach presented in [22], by stating the tackled problem as well as discussing pros and cons, patterns support selection of adequate strategies during process enactment that is problem-driven dynamic process tailoring.

A difference to other existing approaches is the explicit modeling of relationships between work products, modeling concepts, and notations, allowing us to describe and integrate generic processes, referring to work products only in general, with specific development processes providing concrete modeling concepts and notations.

To realize a process pattern approach within a company software developers need some guidance to find their way through the vast number of process patterns that may have been developed at their company. Moreover due to the continuous evolution and change of a living development process patterns a tool to store, present and manage process patterns and work product definitions would be very desirable. We are working on a tool supporting process model maintenance. By realizing the presented pattern maps this tool is supposed to provide guidance for software developers in finding their way through the jungle of process patterns.

To sum up the application of a process pattern approach seems to be very promising, as it provides a flexible way to define a tailored development process that can be easily adapted to new requirements. Combined with a reasonable tool support for the management and development of process patterns this approach may help organizations to create and evolve their custom development process.

References

1. Dirk Ansoerge, Klaus Bergner, Bernd Deifel, N. Hawlitzky, C. Maier, Barbara Paech, Andras Rausch, Marc Sihling, Veronika Thurner, Sascha Vogel. Managing Componentware Development - Software Reuse and the V-Modell Process. In Lecture Notes in Computer Science 1626, Advanced Information Systems Engineering, Page 134-148, Editors Matthias Jarke, Andreas Oberweis. Springer Verlag. 1999.
2. Scott W. Ambler. Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press. 1998.
3. Scott W. Ambler. More Process Patterns: Delivering Large-Scale Systems Using Object Technology. Cambridge University Press. 1999.
4. Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley. 1999.
5. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. Pattern-Oriented Software Architecture, A System of Patterns. John Wiley & Sons.. 1996.
6. Barry Boehm. A Spiral Model of Software Development and Enhancement. ACM Sigsoft Software Engineering Notes, Vol. 11, No. 4. 1986.
7. Klaus Bergner, Andreas Rausch, Marc Sihling, Alexander Vilbig. A Componentware Development Methodology based on Process Patterns. Proceedings of the 5th Annual Conference on the Pattern Languages of Programs. 1998.
8. Klaus Bergner, Andreas Rausch, Marc Sihling, Alexander Vilbig. A Componentware Methodology based on Process Patterns. Technical Report TUM-I9823, Technische Universität München. 1998.
9. J.-C. Derniame, B. Ali Kaba, D. Wastell (eds.): Software Process, Principles, Methodology, and Technology. Lecture Notes in Computer Science 1500, Springer, 1999.

10. Tom DeMarco, Timothy Lister. *Peopleware, Productive Projects and Teams*, Second Edition Featuring Eight All-New Chapters. Dorset House Publishing Corporation. 1999.
11. Desmond Francis D'Souza, Alan Cameron Wills. *Objects, Components, and Frameworks With Uml: The Catalysis Approach*. Addison Wesley Publishing Company. 1998.
12. Wolfgang Dröschel, Manuela Wiemers. *Das V-Modell 97*. Oldenbourg. 1999.
13. Ivar Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Publishing Company. 1992.
14. Ivar Jacobson. *Component-Based Development Using UML*. Invited Talk at SE:E&P'98, Dunedin, Newzealand. 1998.
15. Ivar Jacobson, Grady Booch, James Rumbaugh. *Unified Software Development Process*. Addison Wesley Publishing Company. 1999.
16. C. Klein, B. Rumpe, M. Broy: A stream-based mathematical model for distributed information processing systems - SysLab system model. In *Proceedings of the first International Workshop on Formal Methods for Open Object-based Distributed Systems*, Chapman & Hall, 1996.
17. Philippe Kruchten. *The Rational Unified Process, An Introduction*, Second Edition. Addison Wesley Longman Inc. 2000.
18. Ingolf Krüger. *Distributed System Design with Message Sequence Charts*. Dissertation, Technische Universität München. 2000.
19. Object Management Group (OMG). *Meta Object Facility (MOF) Specification*. <http://www.omg.org>, document number: 99-06-05.pdf. 1999.
20. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. *Capability Maturity Model for Software, Version 1.1*. Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403. 1993.
21. Winston W. Royce. *Managing the Development of Large Software Systems: Concepts and Techniques*. In *WESCON Technical Papers, Western Electronic Show and Convention*, Los Angeles, Aug. 25-28, number 14. 1970. Reprinted in *Proceedings of the Ninth International Conference on Software Engineering*, Pittsburgh, PA, USA, ACM Press, 1989, pp. 328-338.
22. C. Rolland, N. Prakash. A. Benjamin: A multi-Model View of Process Modelling. *Requirements Engineering Journal*, to appear.
23. Bernhard Rumpe: *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. Herbert Utz Verlag Wissenschaft, 1996.
24. A.-W. Scheer: *ARIS, Modellierungsmethoden, Metamodelle, Anwendungen*. Springer Verlag, 1998.
25. B. Schätz, F. Huber: Integrating Formal Description Techniques. In: *FM'99 - Formal Methods, Proceedings of the World Congress on Formal Methods in the Development of Computing Systems, Volume II*. J. M. Wing, J. Woodcock, J. Davies (eds.), Springer Verlag, 1999.
26. OMG: *Unified Modeling Language Specification, Version 1.3 alpha R5*, March 1999, <http://www.omg.org/>.
27. Workflow Management Coalition: *Terminology & Glossary*. Document Number WfMC-TC-1011, Status 3, www.wfmc.org, February 1999.
28. A. Finkelstein, J. Kramer, B. Nuseibeh: *Software Process Modelling and Technology*. Research Studies Press Ltd, JohnWiley & Sons Inc, Taunton, England, 1994.
29. R. Conradi, C. Fernström, A. Fuggetta, R. Snowdon: *Towards a Reference Framework for Process Concepts*. In *Lecture Notes in Computer Science 635, Software Process Technology*. *Proceedings of the second European Workshop EWSPT'92*, Trondheim, Norway, September 1992, pp. 3-20, J.C. Derniame (Ed.), Springer Verlag, 1992.

A Classification Scheme for Studies on Fault-Prone Components

Per Runeson¹, Magnus C. Ohlsson¹ and Claes Wohlin²

¹ Dept. of Communication Systems, Lund University, Box 118, SE-221 00 Lund, Sweden
{Per.Runeson, Magnus_C.Ohlsson}@telecom.lth.se

² Dept. of Software Engineering & Computer Science, Blekinge Institute of Technology,
SE-372 25 Ronneby, Sweden, Claes.Wohlin@bth.se

Abstract. Various approaches are presented in the literature to identify fault-prone components. The approaches represent a wide range of characteristics and capabilities, but they are not comparable, since different aspects are compared and different data sets are used. In order to enable a consistent and fair comparison, we propose a classification scheme, with two parts, 1) a characterisation scheme which captures information on input, output and model characteristics, and 2) an evaluation scheme which is designed for comparing different models' capabilities. The schemes and the rationale for the elements of the schemes are presented in the paper. Important capabilities to evaluate when comparing different models are rate of misclassification, classification efficiency and total classification cost. Further, the schemes are applied in an example study to illustrate the use of the schemes. It is expected that applying these schemes would help researchers to compare different approaches and thereby enable building of a more consistent knowledge base in software engineering. In addition it is expected to help practitioners to choose a suitable prediction approach for a specific environment by filling out the characterisation scheme and making an evaluation in their own environment.

1 Introduction

In the software engineering research community, various approaches for identification of fault-prone components or modules are published, see for example [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. However, the studies are conducted in many different ways and hence the results are not comparable. The studies compare different aspects using different data sets, leading to inconsistent results and confusion. To enable building of knowledge and experience in software engineering, it is necessary to have methods to support comparison and replication of results. The objective here is to provide such support for predicting fault-prone components.

Most studies are conducted using the same principles. First, metrics are collected to be used as predictors. The data are divided into two sets, *fit data* and *test data*. Second, the prediction model is built using the *fit data* set and some statistical technique. The model is evaluated using the *test data* set, and possibly re-built and re-evaluated. Third, the same metrics are collected in a new project and the prediction model is applied in a

development project. The output from this step is a prediction of fault-proneness that can be used for quality management purposes.

Even though almost all the published studies are conducted according to the same principles, there are numerous variants of the studies. Different metrics are collected, different statistical techniques are used in building the models, different aspects of fault-proneness are output from the models and they are evaluated in different ways. This makes it hard to make cross-study and meta analyses about prediction of fault-proneness. Furthermore, it is hard for a potential user of the prediction approaches to judge which one to choose. E.g. what does it mean to compare a prediction model which classifies a module as non fault-prone if they are fault-prone in 73% of the cases [4], to another model which has a misclassification rate of 25% [8] on another data set? Which is the better in general, and which is the better for a certain context?

In this paper we propose a classification scheme as a first attempt to overcome those hurdles. The scheme has two constituents, 1) a characterisation scheme which summarises general characteristics, characteristics of the input, the output and the prediction approach, and 2) an evaluation scheme which involves quantitative criteria for evaluation of the ability of predicting fault-proneness for the approach. The classification scheme is an initial attempt to address two different purposes. The first purpose is to characterise new approaches being published, thus bringing more consistent knowledge to the software engineering community, e.g. by meta-analyses. The second purpose is for intended users of the prediction approaches, for example software project managers, to help selecting a suitable prediction approach and evaluate it in a specific context.

In this paper, the techniques applied to build a prediction model are referred to as *method*. The *model* itself is the mathematical formula defining the prediction calculations. The combination of a method used for model building and a selected set of metrics are referred to as a *prediction approach*. A specific instance of an approach is referred to as a *study*.

The paper is organised as follows. In Section 2 the characterisation scheme is presented. The different characteristics and example criteria are given. The evaluation scheme is presented in Section 3. In Section 4 the schemes are used in an example evaluation of three different approaches to illustrate the use of the schemes. Finally, the contributions are summarised in Section 5.

2 Characterisation Scheme

In the strive for managing software projects and in particular the quality of software being developed, different approaches have been defined to estimate properties of software, related to faults resided in the product. The approaches use different metrics, different methods and provide different outputs. In order enable comparison between different approaches, we propose a characterisation scheme.

The scheme contains characteristics that are subject to choices when selecting an approach to apply in a particular environment. The items in the scheme are not orthog-

onal: a choice of a certain value of a characteristic may limit the choices for other characteristics, i.e. the choices are nested. However, the characteristics included in the scheme are those on which decisions are to be taken by the users of an approach. The characteristics are:

- General – application domain and purpose of approach.
- Output – output type and output metrics.
- Input – artefacts and input metrics.
- Model – model class, optimization criterion and method.

The characteristics chosen are presented in the sections below and summarised in a scheme in Table 1.

Table 1. Characterisation scheme.

	Item	Value
General	Application domain	Telecommunications, Aeronautics, Automotive, Information systems
	Purpose of study	Identify all/certain share of fault-prone components
Output	Output type	Estimation Ranking Classification
	Output metric	Number of faults (per unit) Order of fault-proneness Fault-prone / non fault-prone
Input	Artefact	Requirements specification Design specification Code Test specification
	Input metrics	List of metrics (requirements, design, code, test)
Model	Model class	1: Input (Product) → Output (Product) 2: Input (Product, Resource) → Output (Product, Resource) 3: Input (Product) → Output (Process) 4: Input (Process) → Output (Process)
	Optimization criterion	Misclassification Cost of misclassification Total classification cost
	Method	Principal Component Analysis Boolean Discriminant Analysis Spearman Rank Correlation Optimized Set Reduction Neural Network Regression Analysis ...

2.1 General

The general characteristics present which application domain the models is applied to and the purpose of the study. The purpose is in most cases to identify fault-prone components to enable quality improvement activities. The differences may be in the quality requirements, whether all faults should be removed which is typical for an aeronautics application, or if some level of fault density is acceptable, as the case in telecommunications.

2.2 Output Type

In this characterisation, we are primarily focused on the applicability of the approaches; thus we begin with the output. There are three principal types of output:

1. Approaches for *estimation* of fault content. These approaches try to estimate the number of faults resided in a software component or other artifact. An example of an approach is presented by Ming et al. [9].
2. Approaches for *ranking* fault-prone components. These approaches rank the components with respect to their predicted fault-proneness, and enable classifying components as fault-prone according to a threshold. For example, they may pick out the 10% most fault-prone components. An example of an approach is presented by Ohlsson and Alberg [6] and further refined by Ohlsson and Wohlin [7].
3. Approaches for *classifying* fault-prone components. These approaches classify components into either fault-prone or non fault-prone based on some fixed threshold criterion, for example pointing out all components which have more than five faults. Examples are presented by Khoshgoftaar et. al. [4] and Schneidewind [8].

The output type is hence *estimation*, *ranking* or *classification*. The classification is the weakest form of model. A ranking model can be transformed into a classification model by cutting the ranking list into two, based on some criterion, e.g. percentage of components. An estimation model can be turned into a ranking model by sorting the components according to their estimated value, and turned into a classification model by comparing a threshold on the estimated value.

2.3 Output Metrics

The metric, which is produced as an output of the estimation model, is mainly limited by the output type, as defined above. However, minor variants exist:

- Estimation approaches may predict the number of faults per component, the number of fault per line of code or faults per some other length or size metric.
- The output metric from classification approaches is a boolean; fault-prone or non fault-prone or three-level [12]
- Ranking approaches provide an order number for each component with respect to its fault-proneness.

The metrics are all product metrics, i.e. characteristics of the product itself [11].

2.4 Artefacts

The next characteristic concerns the artefacts to which the approach is applicable. A typical use for a fault-proneness prediction approach is to predict, based on data from one type of artefacts, which artefacts are likely to be fault-prone in a subsequent phase. A quality improvement effort is then planned in the subsequent phase based on the prediction result. For example, more testing time may be planned to fault-prone components than to other components based on data from a design document.

The artefact characteristic tells to which artefacts the approach may be applied. Here we have chosen to illustrate the artefacts through a standard setup of documents: Requirements specification, Design specification, Code, Test specification. The published approaches are primarily applied to the design and code artefacts.

2.5 Input Metrics

The characteristics defined so far, define which prediction we want – the output – and to which type of documents we want to apply the prediction – the artefact. Now we have to study which type of information is available as input to the approach.

The input metrics are the list of metrics collected and used to build the prediction model and to predict fault-proneness of components. The choice of metrics is primarily restricted by the artefact to which the approach is applied. For example, lines of code metrics are not available before the code is available and can hence not be used for prediction purposes if the approach is to be used in the design phase. Most approaches are based on product metrics, either from design or code documents.

2.6 Model Class

Given the input and output metrics, the resulting prediction model can be classified according to the scheme proposed by Fenton and Pfleeger [11]. The classes do not cover all possible combinations of inputs and outputs, but the feasible ones, according to Fenton and Pfleeger.

- Class 1: Input (Product) → Output (Product)
- Class 2: Input (Product, Resource) → Output (Product, Resource)
- Class 3: Input (Product) → Output (Process)
- Class 4: Input (Process) → Output (Process)

As most models use product metrics from artefacts in earlier phases and predicts products metrics of artefacts in later phases, we generally have models of class 1.

2.7 Optimization Criterion

When fitting a model to the fit data, we may optimize the model according to a specific criterion. This far, we have seen three different criteria published:

- *Misclassification.* This means that the model is Optimized to produce as few misclassifications as possible. A misclassification means that a fault-prone component is classified as non fault-prone or vice versa.
- *Cost of misclassification.* It is generally accepted that some misclassifications cost more than others do. It is more costly to classify a fault-prone component as non fault-prone, which means that faults are slipping through into later phases implying more expensive identification and repair. Classifying non fault-prone components as fault-prone means wasting effort in the project.
- *Total classification cost.* This means that the model is Optimized to give as low cost as possible. The cost included is the cost of the quality actions spent on the components correctly classified as fault-prone, and the cost of misclassification.

The optimization criterion may be one of the three described above.

2.8 Method

Finally a statistical or another method is applied to build the prediction model. Methods utilised include, for example, Principal Component Analysis (PCA) [3], Boolean Discriminant Analysis [8], Spearman Rank Correlation [5], Optimized Set Reduction [1], Neural Network Techniques [2] and Regression Analysis [9]. There are possibly other methods that can be used.

2.9 Use of the Classification Scheme

The classification scheme is intended to bring clarity to differences and similarities between different approaches. To the practitioner, the input and output sections are of primary interest, while for the researcher, the model section is of interest as well.

3 Evaluation Scheme

Among the approaches published in the literature for predicting fault-proneness, many are evaluated with respect to their ability to predict properly. However, there are almost as many evaluation criteria as there are prediction approaches. In order to enable clearer comparisons and cross-study evaluation, we propose an evaluation scheme to be used when evaluating the performance of prediction approaches. Furthermore, detailed enough descriptions of the approaches should be published to enable true evaluation and replication.

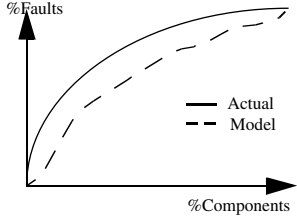
The evaluation criteria are put together in a scheme, in Table 2. Part a) contains a contingency table to show an overall picture of the estimation ability of the approach. Part b) is applicable only to the ranking type of approaches, showing an Alberg diagram [6]. The concept of the Alberg diagram is that if the components are placed in decreasing order according to the number of faults, then the accumulated number of faults for different percentages of the components would be the best any model could achieve. By comparing the actual and the model curves the quality of the model is quantifiable. In the evaluation scheme, we propose criteria with respect to *overall misclassification*, *classification efficiency* and *total classification cost* instead of focusing only on one of them. Part c) is a definition of estimation accuracy for estimation type models.

3.1 Misclassification

The evaluation criterion relates to the approaches' ability to point out components as fault-prone and non fault-prone. Here we define two types of errors that may be conducted when trying to predict fault-prone components¹:

-
1. Sometimes the opposite definitions are used, see for example [10], which depends on which hypothesis is tested. We define the null hypothesis, H_0 : The component *is* fault-prone, and the alternative hypothesis, H_1 : The component is *not* fault-prone.

Table 2. Evaluation scheme for a) classification models, b) ranking models and c) estimation models.

Classification	a)	Model non fault-prone	Model fault-prone	Total	Ranking	
	Actual non fault-prone	A	B Type II	A+B		
	Actual fault-prone	C Type I	D	C+D		
	Total	A+C	B+D	A+B+ C+D		
	Type I rate $C/(C+D)\%$ Type II rate $B/(A+B)\%$ Overall misclassification $(B+C)/(A+B+C+D)\%$ Classification efficiency $(B+D)/(C+D)$ Total classification cost: $C * \text{Cost}_{\text{Type I}} + (B+D) * \text{Cost}_{\text{Type II}}$				Estimation	c) <i>Accuracy</i> $= \frac{\text{Faults}_{\text{Estimated}} - \text{Faults}_{\text{Actual}}}{\text{Faults}_{\text{Actual}}}$

- Conducting a type I error means classifying actually fault-prone components as non fault-prone.
- Conducting a type II error means classifying actually non fault-prone components as fault-prone.

The rate of the two error types as such has been used as an evaluation criterion [3, 8]. The approach, which minimises the misclassifications, is then considered the best. The overall misclassification is the total number of the misclassifications of type I and II, normalised by the total number of components. The criteria are defined as follows:

$$\text{Type I rate} = C/(C+D)\% \quad (1)$$

$$\text{Type II rate} = B/(A+B)\% \quad (2)$$

$$\text{Overall misclassification} = (B+C)/(A+B+C+D)\% \quad (3)$$

3.2 Cost of Misclassification

The number of errors does not capture the cost aspects of the classification. Type I errors are, for example, in general more expensive to commit than type II errors are. A fault that remains in a product in the operational phase, costs more than the effort spent on, for example, reviewing an additional component which is misclassified as fault-prone but it is actually not. Therefore the misclassification cost (MCC) is introduced as an evaluation criterion [10], here presented with our notations:

$$MCC = C * C_{\text{Type I}} + B * C_{\text{Type II}} \quad (4)$$

It shall be noted that all costs used in this paper are relative costs. In [10] the ratio between $C_{\text{Type I}}$ and $C_{\text{Type II}}$ is exemplified with a factor 10.

The problem with this evaluation criterion is that it, in many cases, has its optimum driven by the type I error. This implies that too many components are pointed out as

fault-prone to minimise the risk for type I errors. This provides low type I error rate but a high type II error rate, since the cost of type II errors is lower than the cost of type I errors.

The main purpose of classification approaches is to optimize the use of resources, i.e. the primary concern is the overall cost spent on identifying and improving the fault-prone components. In other words, we would like to minimise the effort spent in development and in maintenance due to the faults in the software. We refer to this as the Total Classification Cost (TCC), which is defined as:

$$TCC = C * C_{Type I} + (B+D) * C_{Type II} \quad (5)$$

The difference between equations (4) and (5) is the last term which adds the cost of effort spent on components which are correctly classified as fault-prone, thus enabling comparison to the cost of spending effort on all components. The TCC shall be as low as possible. In any case, it shall be less than spending additional effort on all components, i.e. $(A+B+C+D)*C_{Type II}$.

For ranking approaches, the evaluation criteria presented above have to be calculated for each threshold value chosen. Instead of tabulating all these data, we propose the Alberg diagram to be used [6]. The Alberg diagram is a graphical representation of the approaches' capability of pointing out fault-prone components, see Table 2b.

For estimation approaches, they can be turned into ranking approaches by sorting the components according to their estimated value, and into classification approaches by applying a threshold. Then the Alberg diagram and the contingency table can be used. In addition, the estimation accuracy can be used as an evaluation criterion, specifically for estimation approaches, see Table 2c.

3.3 Classification Efficiency

The classification efficiency is an evaluation criterion related to cost and misclassifications. However, it provides a viewpoint that is of importance from a practical project management perspective. We define the classification efficiency as the number of components classified as fault-prone divided by the actual number of fault-prone components.

$$\text{Classification efficiency} = (B+D)/(C+D) \quad (6)$$

The optimum value of classification efficiency is hence 1 (B and C equal 0). A value less than 1 means that too few components are classified as fault-prone while a value of more than one means that too many components are classified as fault-prone.

3.4 Use of the Evaluation Scheme

The evaluation scheme is intended to compare capabilities of the different models. For a practitioner, it can be used to compare different methods in a specific environment, while the researcher may compare methods applied to data from different environments. Depending on which of the characteristics is considered most important, the

misclassification, the cost of misclassification or the efficiency is given the highest priority in the comparison.

4 Example Study

In this section we apply the characterisation and evaluation schemes presented in Section 2 and Section 3 to illustrate typical use of the schemes. The example context is an organisation which intends to investigate methods for identification of fault-prone-ness in their specific settings. They have selected three different candidate approaches:

- Schneidewind (S97) [8]
- Khoshgoftaar et. al. (K96) [4]
- Ohlsson and Wohlin (O98) [7]

In Table 3 the results from the characterisation can be found.

Table 3. Characterisation of three example approaches.

	Item	S97	K96	O98
General	Application domain	Aeronautics	Telecommunications	Telecommunications
	Purpose of study	Identify all fault-prone components	Identify a certain share of fault-prone components	Identify a certain share of fault-prone components
Output	Output type	Classification	Classification	Ranking
	Output metric	Fault-prone / non fault-prone	Fault-prone / non fault-prone	Order of fault-prone-ness
Input	Artefact	Code	Code	Design
	Input metrics	List of code metrics, see [8].	List of code metrics, see [4].	List of code metrics, see [7]
Model	Model class	Class 1: Input(Product) -> Output(Product)	Class 1: Input(Product) -> Output(Product)	Class 1: Input(Product) -> Output(Product)
	Optimization criterion	Misclassification	Misclassification	Total classification cost
	Method	Kolmogorov-Smirnov and Boolean Discriminant Analysis	PCA and non parametric Stepwise Discriminant Analysis	Spearman Rank Correlation

4.1 Characterisation

General. The approaches represent two different application domains, aeronautics and telecommunications. The domains also reflect the purpose, where the aeronautics approach aims at identifying all fault-prone components, while in the telecommunications domain, only components with more the a certain number of faults are considered fault-prone.

Output. The approaches represent two output types, *classification* and *ranking*. S97 and K96 both produce a binary result, i.e. fault-prone or non fault-prone, while O98

ranks the components in decreasing order according to their fault-proneness. The preferred output type depends on the objective of the study. The objective for project management might, for example, be to priorities available resources like test time and review time and then S97 and K96 could be considered.

Another case is where management, for example, can spend effort on additional reviews on 20 percent of the components. Which components should be considered? To answer this question, approaches like O98, which rank the components in fault-proneness order, are more useful.

Another difficulty with the classification approaches is which threshold value to choose. Depending on the domain the threshold may differ. For example, the S97 approach was developed to be applied to the aeronautics domain, thus quality requirements are very high and the threshold is set to zero faults in the components [8]. The K96 approach sets the threshold based on heuristics [4], i.e. after a discussion with project engineers they decided on a certain threshold.

Input. In order to enable early planning of actions to prevent fault-proneness, the approaches shall be applicable as early as possible. The phase when the approach is applied and the input metrics depend on each other. If the prediction model is based on metrics from the design phase, it can be applied earlier than if it is based on code metrics from the implementation phase. In this case, the O98 approach is based on design metrics while K96 and S97 are based on code metrics.

Depending on the purpose of the application of the approach, it is feasible to use the prediction model with other metrics than described in the original approach. For example, the S97 model could be used with design metrics. In the example study, we have a set of design metrics available, which are to be used, see Table 4.

Model. All three approaches take product metrics as input and predict product metrics; thus they belong to model class 1 according to Fenton and Pfleeger's classification [11]. The optimization criteria are different for the approaches. S97 and K96 use misclassification while O98 uses total classification cost.

Different statistical methods for model building are used. S97 uses Boolean Discriminant Analysis. To choose suitable metrics, S97 use Kolmogorov-Smirnov distances. K96 uses Non-parametric Discriminant Analysis and Principal Components Analysis to extract factor scores to be used in the Discriminant Analysis. Finally, O98 uses Spearman Rank Correlations.

4.2 Evaluation

To be able to compare the different approaches in the example we use the same data set for all candidates even though they were originally created for use in a different phase or with some other input metrics. Hence, actually only the prediction method are used, not the complete approach (see Section 1). This is the typical situation when using the method in a new context where different sets of metrics are collected.

Data Description. The example study is based on an investigation of 28 software components for two releases of a large real time system in the telecommunication domain. The actual releases investigated are considered typical for releases of this particular system, although the choice of system and releases was based on availability. The data set has been used earlier in other studies, for example [12].

The size of the 28 components varies between 270 and 1900 lines of code. The programming language used is a company internal language, targeted at their specific hardware platform. For each of the components, the number of fault reports from test and operation were counted and several design and code metrics were collected.

In total, ten metrics originating from the design documentation were collected. The data collected were primarily a count of the number of symbols of different types. The language used during design is a predecessor to SDL, (ITU-T Specification and Description Language) [13], and it is basically a finite-state-machine with states and signals. A modified version of McCabe's Cyclomatic Complexity [14] was used as well. The modification was simply due to being able to handle signal sending and reception respectively. The design metrics are listed in Table 4. Please note that all metrics are scaled due to confidentiality.

Table 4. Design metrics collected.

Metric	Description
SDL-pages	Number of design pages in terms of SDL diagrams.
Tasks	Number of task symbols in the SDL descriptions.
Outputs	Number of output symbols in the SDL descriptions.
Inputs	Number of input symbols in the SDL descriptions.
If	Number of if-boxes in the SDL descriptions.
States	Number of state symbols in the SDL descriptions.
McCabe	This metric is an adaptation of the original proposal by McCabe. The metric is adapted to enable measurement on design and in particular to include the signalling concept used in SDL.
External inputs	Number of unique external inputs to a component.
External outputs	Number of unique external outputs to a component.
Internal signals	Number of unique internal signals within a component.

We defined the fault-prone group as those components with $Faults \geq 10$ and the non fault-prone ones as $Faults < 10$. We used the data from release n as the *fit data* set and the data from release $n+1$ as the *test data* set. In the fit set there were six components classified as fault-prone and in the test data set there were eleven.

In the cost model, we have used a cost ratio between type I and type II errors of 10 (see [10]) i.e. type I errors are ten times as expensive as type II errors. However, to highlight the sensitivity of the results to the cost ratio, a cost ratio of 5 has also been used. The total classification cost based on the lower cost ratio is denoted in parentheses in the tables below.

Results. The results from the application of the *S97* model can be found in Table 5. The classification is based on only one variable, External inputs, The results from the Kolmogorov-Smirnov test provides a maximum difference of 0.818 and a significance of 0.004 for External inputs. To add extra variables when creating the model from the fit data set did not improve the results.

Table 5. Results from *S97*.

Classification		Model non f-p	Model f-p	Total
	Actual non f-p	14	3	17
	Actual f-p	4	7	11
	Total	18	10	28
Type I rate: 36% Type II rate: 18% Overall misclassification: 25% Classification efficiency: 0.9 Total classification cost: 50 (30)				

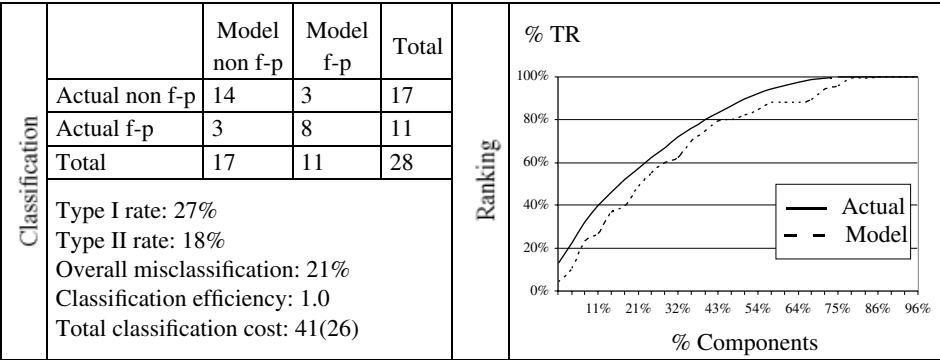
Table 6. Results from *K96*.

Classification		Model non f-p	Model f-p	Total
	Actual non f-p	15	2	17
	Actual f-p	8	3	11
	Total	24	5	28
Type I rate: 73% Type II rate: 11% Overall misclassification: 36% Classification efficiency: 0.45 Total classification cost: 85 (45)				

The results from the *K96* model can be found in Table 6. The Principal Components Analysis provides two factors that are reduced to one when the Stepwise Non-parametric Discriminant Analysis is applied with a significance level of 0.05. The prior probabilities for the fault-prone and the non fault-prone components are 0.79 and 0.21 according to the size of the groups and the smoothing parameter for the normal kernel is 0.05. Choosing the smoothing parameter is difficult and in this case we decided on 0.05, because a larger value would not have classified any components as fault-prone.

The results from the *O98* model can be found in Table 7. The results from the Spearman Rank Correlations provides correlation between External inputs and faults of 0.645 with a significance of 0.001. Though, the combination of If and External inputs had a correlation of 0.677 with a significance >0.001. Therefore, this combination was used.

Table 7. Results from *O98*.



The O98 model has to be evaluated for a given threshold, since it provides a ranking of components in fault-proneness order. In this case we have chosen a threshold of 39 percent, which corresponds to the number of components considered fault-prone for the S97 and K96 models (11/28), thus making the prerequisites the same for all three models. This is shown in the Alberg diagram in Table 7. Noticeable is that the classification efficiency is 1.0 per definition, since the ranking model points out 39 percent of the components, if 39 percent are considered fault-prone. Therefore, the number of type II errors are always equal to the number of type I errors in this model, while the type I and type II *rates* differ.

Interpretation. The scheme contains evaluations with respect to three aspects. They are the following:

- Overall misclassification
- Classification efficiency
- Total classification cost

As mentioned before, we have used a cost ratio between type I and type II errors to be 5 or 10. The different classification results from the previous section are summarised in Table 8, where the reference costs for spending no effort and spending extra effort on all components are provided in the last two rows. Figures from the original studies are presented as a reference in Table 8.

Table 8. Comparison of models in the example study and in the original studies.

	Example study			Original studies		
	S97	K96	O98	S97	K96	O98
Type I rate	36%	73%	27%	2.3%	35.7%	24%
Type II rate	18%	11%	18%	47%	10.5%	58%
Overall misclassification	25%	36%	21%	28%	17.2%	34%
Classification efficiency	0.9	0.45	1.0 ^a	1.6	0.97	1.0 ^a
Total classification cost	50 (30)	85 (45)	41 (26)	79 (74)	129 (79)	516 (321)
Cost with no effort	110 (55)			430 (215)	280 (140)	670 (335)
Cost with extra effort on all	28 (28)			100 (100)	133 (133)	232 (232)

a. By definition

In this example study, the S97 model classified 18 percent as fault-prone even though they were not and the model missed 36 percent of the fault-prone components. The overall misclassification was 25 percent and the classification efficiency was 0.9, i.e. the model classified 0.9 components as fault-prone for every actual fault-prone component. The total classification cost was 50 (30 if the ratio was set to 5). It should be noted that these numbers should be compared to the cost of spending extra effort on all components, which is 28.

The approach in K96 provides a type I rate of 73 percent and a type II rate of 11 per-

cent. The type I rate is very high and the reason is that the model classifies too few components as fault-prone. The model should have predicted at least twice as many component as fault-prone to be able to perform better. Because of the high type I rate, the total classification cost is as high as 85 (45 if the ratio is set to 5).

The O98 model is, as mentioned earlier, a little bit different since it ranks the components instead of classifying them. By definition it classifies the correct number of components and therefore the classification efficiency is 1.0. The type I rate is 27 percent and the type II rate is 18 percent.

The approaches above show different abilities with respect to the evaluation criteria to illustrate the use of the evaluation scheme. An approach should provide estimates with low misclassification rates. If we are to accept errors, type II errors are preferred, since they are less costly than type I errors. S97 and O98 are fairly equal but K96 have a large type I rate. If we are interested in ranking models, O98 should be selected, otherwise either O98 or S97 can be used.

5 Summary

Numerous approaches to prediction of fault-proneness have been presented in the literature. They are evaluated using various schemes and criteria. In this paper an initial attempt to improve the comparison an use of classification schemes is presented. It has two core constituents is presented: 1) a *characterisation* scheme which provides an overview of the input to the approaches, the output and the prediction model itself; 2) an *evaluation* scheme which enables qualitative evaluation of the approaches' prediction capabilities. If these schemes are used in presenting new approaches and evaluation of existing ones, the research progress can be easier to evaluate and apply.

The characterisation scheme can not only be used for presenting studies, but also selecting prediction approaches based on needs by the users, for example software managers. They can identify their needs based on the different characteristics of the approaches and evaluate the capabilities of different approaches in their environment.

To illustrate the usage of the schemes an example study is presented which evaluates three different approaches with different characteristics. For the evaluation scheme we use data from two releases of a system to illustrate the models' abilities to classify components with respect to misclassification rates and costs. The results from the example study are compared to the original studies.

It is concluded that it is not sufficient to compare the misclassification rate. The classification efficiency and total classification cost provide other aspects of the models abilities, which are related to the intended use of the models in a software development context. Furthermore, the example study and the original studies show different results as they are based on different data sets.

The proposed classification scheme would allow for a better comparison between different prediction approaches. The scheme is expected to support comparison, replication of results and meta-analyses, leading to better and more consistent knowledge and more collected experience in software engineering.

Acknowledgment

We would like to thank Per-Erik Isberg, Department of Statistics, Lund University, for his support on statistical methods. This work was partly funded by The Swedish National Board for Industrial and Technical Development (NUTEK), grant 1K1P-97-09690 and 1K1P-97-09673.

References

1. Briand, L.C., Basili, V.R. and Hetmanski, C.J., "Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components", *IEEE Transactions on Software Engineering*, 19(11), 1993, 1028–1044.
2. Khoshgoftaar, T.M. and Lanning, D.L., "A Neural Network Approach for Early Detection of Program Modules Having High Risk in the Maintenance Phase", *Journal of Systems and Software*, 29(1), 1995, 85–91.
3. Khoshgoftaar, T.M., Allen, E.B., Kalaichelvan, K.S. and Goel, N., "Early Quality Prediction: A Case Study in Telecommunications", *IEEE Software* (January 1996), 65–71.
4. Khoshgoftaar, T.M., Allen, E.B., Halstead, R. and Trio, G.P., "Detection of Fault-prone Software Modules During a Spiral Life Cycle", *Proceedings of the International Conference on Software Maintenance*, (Monterey, California, USA, November 96) 69–76.
5. Ohlsson, N., Helander, M. and Wohlin, C., "Quality Improvement by Identification of Fault-Prone Modules using Software Design Metrics", In *Proceedings Sixth International Conference on Software Quality*, (Ottawa, Ontario, Canada, 1996), 1–13.
6. Ohlsson, N. and Alberg, H., "Predicting Fault-Prone Software Modules in Telephone Switches", *IEEE Transactions on Software Engineering*, 22(12), 1996, 886–894.
7. Ohlsson, N. and Wohlin, C., "Experiences of Fault Data in a Large Software System", *Information Technology Management: An International Journal*, 2(4), 1998, 163–171.
8. Schneidewind, N. F., "Software Metrics Model for Integrating Quality Control and Prediction", in *Proceedings Fourth International Software Metrics Symposium*, (Albuquerque, NM, USA, November 1997) 402–415.
9. Zhao, Ming, Wohlin, C., Ohlsson, N. and Xie, Min, "A Comparison between Software Design and Code Metrics for the Prediction of Software Fault Content", *Information and Software Technology*, 40(14), 1998, 801–809.
10. Khoshgoftaar, T.M. and Allen, E.B., "The Impact of Cost of Misclassification on Software Quality Modeling", in *Proceedings Fourth International Software Metrics Symposium*, (Albuquerque, NM, USA, November 1997), 54–62.
11. Fenton, N.E. and Pfleeger, S.L., *Software Metrics: A Rigorous and Practical Approach*, Thomson Computer Press, 1996.
12. Ohlsson, M.C. and Wohlin, C., "Identification of Green, Yellow and Red Legacy Components", In *Proceeding of International Conference on Software Maintenance*, (Bethesda, Washington D.C, USA, November 1998), 6–15.
13. ITU, "Recommendation Z.100: SDL – Specification and Description Language", 1988.
14. McCabe, T. J., "A Complexity Measure", *IEEE Transactions on Software Engineering*, 4(2), 1976, 308–320.

Program Understanding Behavior during Estimation of Enhancement Effort on Small Java Programs

Lars Bratthall¹, Erik Arisholm², and Magne Jørgensen²

¹ ABB Corporate Research, PO Box 90, N-1375 Billingstad, Norway
lars.bratthall@no.abb.com

² Dept. Informatics, University of Oslo, Oslo, Norway
{erika, magnej}@ifi.uio.no

Abstract. Good effort estimation is considered a key success factor for competitive software creation services. In this study, task level effort estimation by project leaders and software designers have been investigated in two Internet software service companies through an experiment. Protocol analysis of 27 think-aloud estimations of effort required for consecutive change tasks on a small Java program have been analysed, using the AFECS coding scheme. Results indicate that a) effort estimation at the task level is very different depending on the individual, even when small problems are addressed; b) AFECS seems to be appropriate to use as a coding scheme when assessing program comprehension behaviour for the purpose of effort estimation; c) protocol analysis of comprehension during effort estimation does not necessarily capture all process elements. These results can be used to further guide detailed analysis of individual task level effort estimation, as can a set of high-level estimation events indicated in this study.

1 Introduction

Software development and enhancement effort estimation models have been developed at least since 1966 [7]. A variety of models are available, see e.g. [3] for an overview. In spite of this the software industry is well known for its inaccurate and over-optimistic effort estimates, leading to impaired quality and delivery delays. A recent survey by the Standish Group [23] found that about one third of the projects investigated had more than 50% cost overrun, compared to the original estimate. It is difficult to plan the market introduction of software intensive products when the software development effort is inaccurately estimated, and, not surprisingly, [23] found that about one third of the projects had more than 20% time overrun. This implies one extra month on a six months project. This delay can be the difference between a success and a failure in some markets. For example, being only one week late to market can make a telecom consumer device fail entirely in the market [2].

A potential reason for the lack of success in the estimation of software development schedule and effort is that most of the research focus has been on *replacing* expert judgement with formal estimation models, not on *aiding* human expert judgement. The high number of estimation models and the low number of studies of expert esti-

mates support this view. According to [5], psychological research on real-world quantitative estimation “has not culminated in any theory of estimation, not even in a coherent framework for thinking about the process”. To the best of our knowledge, there is no in-depth study of how software professionals estimate software development and enhancement effort. The most relevant studies may be those on program comprehension, e.g. [10, 13, 14, 18, 22], and studies comparing the precision of expert estimates compared to model based estimates, e.g [9, 20]. We believe that comprehension processes for the purpose of *estimation* may be different from the comprehension processes studied, and that the studies comparing the estimation precision to expert models provide limited insight into the underlying expert estimation strategies.

Without knowledge about the strategies, the strengths and weaknesses and the individual variance in expert estimation, we are not able to properly combine formalized knowledge, e.g., statistical estimation models, with expert’s intuition-based estimates. For this reason, the objective of this paper is to provide a first step in increasing our knowledge of expert estimation processes.

In order to pursue this objective, an experiment has been conducted among eleven participants from two companies that deliver internet-related software. The participants’ experience range from little to several years of estimation experience. The use of software professionals as opposed to students in the experiment, is important to ensure that the estimation processes studied are representative for those used in parts of the software industry. The estimation tasks studied may be described as: Estimation of effort needed to implement given new requirements on small, previously unknown object-oriented programs. The advantage of this is that all experiment participants had similar domain and program knowledge. The type of programs are considered to be representative for small systems, or modules of larger systems, developed or enhanced in the type of company studied.

First, each experiment participant was interviewed to get an overview of their current project estimation practises. They were then required to estimate how long time three given change requests to a given piece of software would take, while verbalizing their thoughts during the estimation. The transcriptions of the verbalizations are analysed in this study. The method used is described in detail in Section 2. Collected data is presented in Section 3 and further analysed in Section 4. Finally, the study is summarized in Section 5.

2 Experiment Method

In this section, the experiment method used in this study is presented. In Section 2.1, the operations of the experiment is outlined. In Section 2.2, the analysis of the think-aloud sessions is described. In Section 2.3, threats to the validity of this study are discussed.

2.1 Operation and Instruments

When first meeting an experiment participant, anonymity and feedback were assured. The exact purpose of the experiment was presented as “look into effort estimates in your organization”. After this, the participant was asked to fill in an introduction questionnaire. The questionnaire and other instruments used are described in detail in Table 1. Information regarding the participants is shown in Table 2.

Table 1. Instruments used in experiment

Instrument	Description
Introduction questionnaire	A short questionnaire assessing the background of the participant. The purpose was to get input to interviews and to be a primer, i.e. a “mind opener”. Simple questions were used.
Interview guide	The interview guide specified the order of a number of topics to discuss. Some questions were formulated in advance, but additional questions have been allowed. The questions asked are believed not to affect the participants’ responses to later data collection.
Mocca overview	A few pages of overview of some base classes in the Mocca language. Mocca is Java, except that some advanced constructs have been omitted. Those are not present in any code presented to the participants. The purpose of this is to avoid that differing language proficiency impacts the results. The overview includes a small rehearsal task which requires writing about 10 lines of Mocca. Mocca is further described in [1].
Calibration source code	A Mocca program describing an ATM.
Calibration task	A task that describes a change request for the calibration source code. The task is to add a new service: A list of transactions for each account should be made available. The purpose of the task is reduce the threat of having differing experience of Java, as well as the threat of there being a long time since it last was used.
Introduction to think aloud	The example from [8]: “What is 25*36?” First, the experimenter answered the question while verbalizing his thoughts, then the participant was asked to verbalize thoughts while answering “What is 35*31?”.
Think-aloud: Instruction, Tasks	This instruction explained that some estimation tasks should be completed. The tasks were the same for all participants, and delivered in same order, as this allows taking learning effects into account. Each task was to estimate the time it would take to accomplish a change request (expressed as informal text) to the given source code. The tasks included an example of the expected resulting output, after changes. An example of a task, here expressed shortly, is “The coffee machine software currently does not produce hot chocolate. Add functionality so that hot chocolate at the cost of 10 cents can be delivered.”. There were no quality requirements on

Instrument	Description
	the quality of code changes (maintainability, readability, fault tolerance etc.).
Think aloud form	A form where the effort estimates for each change task was to be recorded.
Think aloud source code	To reduce mono-instrument bias, there were two versions of the source code that implemented functionally identical simple coffee machines, design A and design B. Design A is a simple, naive design consisting of 4 classes. Design B is more complex; it is responsibility driven and contains 8 classes. The systems are further described in [1]. In [1] it was identified that on average, the change tasks in this experiment took longer to perform on design B, though less structural changes were needed. The participants were now aware of the existence of two different designs.
Final inter-view guide	The purpose of this interview was to assess the impact of thinking aloud, the external validity of the think aloud source code and the estimation tasks.
Coding scheme	The coding scheme is the same as the one presented in [11], with a few extensions for explicit coding of when estimates are made. The higher hierarchical levels in [11] are not affected at all. The coding scheme is further described in Section 2.2.

Table 2. Participant information. Participants P2 and P8 do not participate in think-aloud estimation

Id	Com-pany	Role	Estimation Experience	Design	MSC Experience
P1	C1	Software Designer	Yes	A	Yes
P2	C1	Project Manager	Yes		
P3	C1	Software Designer	Yes	A	No
P4	C1	Software Designer	Yes	A	Yes
P5	C1	Software Designer	Little	B	Yes
P6	C1	Software Designer	Little	A	Yes
P7	C1	Software Designer	Yes	B	No
P8	C2	Software Designer	No		
P9	C2	Project Manager	Yes	B	Yes
P10	C2	Software Designer	Little	A	Yes
P11	C2	Software Designer	No	B	No

The introduction questionnaire was then used as one input to an interview of the participant, regarding their current project estimation practises.

After the interview, the participants were given ten minutes to study the Mocca programming language and complete a small programming task (about ten lines of programming). After having studied Mocca, all participants were required to complete a calibration task. The purpose was to reduce the impact of differing Java experience. The task was to modify a Mocca program that describes the operation of an ATM, so

that it could show transaction information, such as the latest withdrawals on an account.

Next, the participants were introduced in how to think aloud by using a standard example from [8]: “What is 26×35 ?”. The remaining part of the experiment session was recorded on audio-tape.

The following information was presented:

- An overall instruction regarding what to do, i.e. perform estimates while thinking aloud.
- Three ordered change tasks for which to make effort estimates. Each estimate should include the effort needed for the code change and an individual inspection of the change.
- A sequence diagram (MSC) showing the current operation of a small Mocca program, that controls the operations of a pay-per-cup coffee machine.
- The corresponding Mocca code.

The ordering of the change requests was the same for all participants, as the objective of the study was to simulate a situation when consecutive change requests come from a customer to a development organization. The participants were explicitly instructed to take any learning effects into account. The participant could use any of the instruments available. The switches from e.g. looking at code to looking at the sequence diagram were recorded.

If the participant became silent for more than 10-15 seconds, the researchers asked what the participant was doing. Questions have been asked in a way that suggests the need for more information, using phrases such as “What are you looking at right now?” and “Keep talking.”, instead of for example “*How* do you think right now?”, which could affect the way estimation took place.

After all estimation tasks were finished, the experimenter performed a short interview. The purpose was to establish if the participant believed that the think-aloud process had affected the estimation process. Also, the degree of realism in the task and the original software were assessed.

2.2 Analysis of Think-Aloud Session

The statements uttered during estimation have been written down into a protocol. Each utterance in the protocol has been classified using one or more codes from a coding scheme, AFECS by [11], i.e. protocol analysis has been performed. The ordered set of codes from a protocol is called a transcript.

The purpose of AFECS is to provide an extendable, standardized coding scheme that is “consistent with accepted theories of program comprehension”, i.e. the Integrated Model of Comprehension [15] that has been used in several related studies [12, 13, 16, 17]. Since it was assumed that a large part of an estimation process would be to comprehend the code to change and the change task, AFECS was selected as the coding scheme for analysing of the estimation process. It was believed that the extensibility mechanism would be advanced enough in case the AFECS had to be extended to code utterances related to the particular situation (program understanding *for the purpose of effort estimation*). An example of an AFECS transcript is shown in

Figure 1. The three top levels of the AFECS hierarchical coding scheme is shown in Figure 2.

I want to see how it
processes this template definition,
SYS.gol.not

so I see an if branch and it's
saying, if isTemplate infile.
SYS.act.inf.cod.rea.lvl

Lets see I'm not sure what infile is,
so I'm going to highlight infile,
just to see where, where it is used,
OPP.act.inf.knw.rev
OPP.gol.not
SYS.act.inf.cod.sea.var

Fig. 1. Example of AFECS transcription from [11]

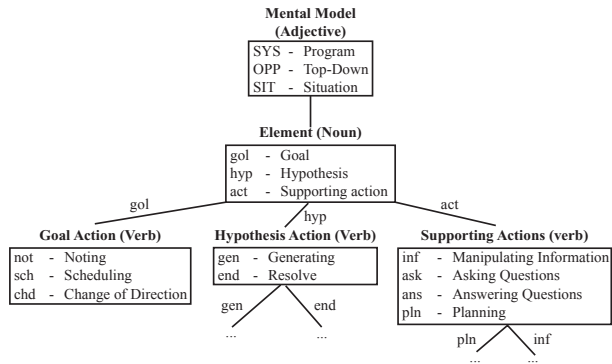


Fig. 2. Part of AFECS coding scheme [11]

The hierarchical structure of AFECS allows analysis at different level of detail. In this study, the focus is on the three top level mental models coded; SYS, OPP and SIT. The different models are discussed in depth in e.g. [11, 15] and are here only briefly described.

- **SYS:** The *program model*. When code is new or unknown, programmers first build a control flow abstraction called the program model.
- **OPP:** The *top down (or domain) model*. This model of program understanding is usually used if the code or the type of code is familiar. For example, a domain model of a coffee machine would include knowledge of what components (or modules) such a machine can have, and how they interact.
- **SIT:** The *situation model*. The situation model is a data-flow/functional abstraction. It is developed using the program model.

AFECS allows fine grained analysis of the estimation process. In order to capture higher-level events, such as e.g. the forming of an initial (called *primary* hypothesis by [4]) top-level hypothesis regarding what to change, automatic process discovery [6] through detections of patterns consisting of multiple codes in a transcript was initially considered. However, the higher-level patterns seemed to have very different sets of codes. Sometimes, higher-level events even seemed to be interleaved. Therefore manual detection of higher-level patterns has been used, at the cost of lower reliability in this study due to possible researcher bias.

Which higher level patterns to look for was determined by reading the set of protocols multiple times. From this, a set of higher-level events gradually emerged. When this set seemed stable despite iterating reading the protocols, it was determined that the set of high-level events was complete enough to use for further analysis.

2.3 Threats to Validity

The validity of the findings reported depends to a large extent on how well threats have been handled. Four types of validity threats [26] are analysed: Conclusion validity, construct validity, internal validity and external validity.

Conclusion validity concerns the relationship between the treatment and the outcome. All instruments have been reviewed in advance, and a pilot run of the entire experiment has been made in advance. Many of the instruments have also been used in [1]. The coding table has been used before as described in [11]. The single encoder in this study has used examples of transcriptions in that paper to ensure that the coding is as similar as possible to that in [11]. No other tests investigating if coding is made similar across studies have been performed. However, in [11] there is 98% coder/reviewer-agreement on coding. This builds some confidence in that the codes used are defined clearly enough to produce comparable results also across studies. The use of forms and written instructions, and the fact that the same experimenter has been present during all data collection sessions should ensure that treatments are implemented in the same way for all participants. The group of participants is rather homogeneous as reflected in Table 2. This can affect external validity, but it is believed that the participants are representative for the type of organizations studied. The set of higher level events verbalized has not been validated through other studies.

Internal validity concerns matters that may affect an independent variable's causality, without the knowledge of the researcher. As the entire duration of the experiment is two hours, there should be no maturation effects. The participants are not aware of any hypotheses. There has been some mortality: Two of the eleven participants did not know Java. Therefore, these have not performed the think-aloud estimation. The participants have been explicitly instructed not to discuss the experiment with other participants, and since the participants does not know of any other treatment than the one delivered to them, there should be no compensatory rivalry or resentful demoralization.

Construct validity concerns whether we measure what we believe we measure. All instruments have been reviewed and tested in advance. Some of the instruments have been used by 54 participants in a previous experiment [1]. Mono-instrument bias should be reduced, since the experiment was performed using two different source codes providing the same functionality. The threat of confounding levels of Java experience (or time since Java was used last) should be lessened by the Mocca overview and by the calibration task. Though it is possible that there is an interaction between the initial interview and the later think-aloud session, care has been taken in the interview guide to avoid questions that may change the way the estimation was performed. Since each participant has been exposed to only one treatment (i.e. design A alternatively design B), there should not be any hypothesis guessing. Through guaranteeing anonymity the risk that people work differently from their normal practise because they believe they are in an evaluation situation should be minimized.

External validity concerns generalization of the findings to other contexts and environments than the one studied. A threat related to the understanding of the provided code, is the wide range of comprehension strategies used by different individuals [14,

24]. The biggest threat to external validity we can think of, is that the Mocca programs are not large compared to industrial practise. However, in [1] it is argued that the programs exhibit a wide range of characteristics of larger programs. In the final interview, the participants were asked about the realisticness of the tasks, as well as of the code provided. The general opinion was that the tasks were very realistic given the provided software, but that the program seemed smaller than usual. Some participants also found the code to be of higher quality than usual. However, it was also claimed that the size of code did not affect the way estimation was performed, relatively a commercial project. Thus, we believe that results could be generalized at least to effort estimation for changes in a single module in a larger system. Further research is necessary to fully establish the size limit of programs to which the findings are generalizable.

3 Results – Raw Data

In Sections 3.1-3.2, raw data from the experiment are presented. This data is further analysed in Section 4. In Section 3.3, the validity of data collected is discussed.

3.1 Results Related to the Expressed Estimation Tactics

By iteratively reading the transcripts from the change tasks, the activities in Table 3 have been deducted as occurring during the estimation tasks studied.

Table 3. Description of expressed estimation tactics. An ‘X’ indicates that something is present. A number indicates the number of change tasks in which something is present. Participants P2 and P8 did not participate in the think-aloud estimation. Participant P11 is not present in the table due to that the estimates were “guesstimates” rather than estimates.

Participant action	P1	P3	P4	P5	P6	P7	P9	P10
Sequence diagram related high-level events								
Uses sequence diagram at all	X	X		X	X	X	X	X
Explicitly determines if sequence diagram is conceptual, or shows interactions between classes in code	X				X			X
Uses sequence diagram to understand control flow in software	1			3			2	1
Makes hypothesis regarding what to change from sequence diagram	1			2			2	
Makes hypothesis regarding where to change from sequence diagram	1			2			2	
Code related high-level events								
First look at code: Search for some kind of main functionality.								2
First look at code: Search for a user interface as beacon in code.	2	1	1		1	1		

First look at code: Searches for other beacon in code.	1	2	1	1	1	1	1	1
Uses identifier names as a source of understanding what a piece of software does.	2	2	1	1	22		1	
Uses identifier names to denote a <i>location</i> in code.	2	2	1	1	1	1	1	
Hypothesis for required changes								
Tries to partition the problem before looking at code.	2		1			3		
Once a starting point for changes in code has been identified, follows control flow to check smaller hypotheses generated on-the-fly regarding expected needed changes.	3	3	2	3	3	3	3	1
Double checks own solution by final walk-through	1	1				1		
Generation of estimate								
Compares estimated time to previous estimates							2	2
Uses number of changes in estimation	1	2	1				1	1
Uses complexity of changes in estimation							3	1
Assorted high-level events								
When faced with multiple entities of similar kind, makes a written list	1			2		2	3	3
Expresses lack of design rationale: "What is the thought behind the design?"				1		2		

3.2 Results Related to Thought Process Levels

The results from the AFECS protocol analysis are presented through a number of figures. 27 effort estimates have been performed, resulting in in total 728 codes. In Figure 3, an example transcription is shown. In Figure 4 the average fraction of codes for the SYS, OPP and SIT mental models are presented, as well as the probabilities for proceeding from one type of model to another. In Figure 5, the number of codes of the SYS, OPP and SIT types are shown for each participant. In Figure 6, the number of codes per task and participant is presented.

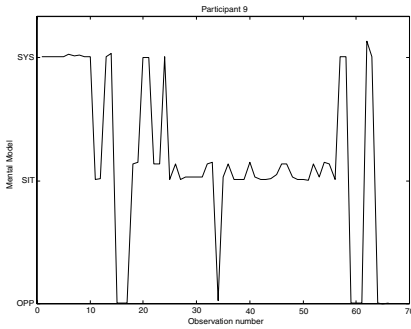


Fig. 3. Example transcription. The small variations around the SYS, OPP and SIT levels are different sub-types of these mental models

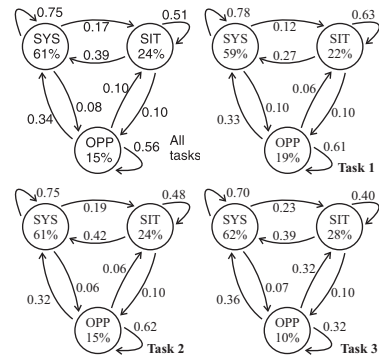


Fig. 4. Average number of events and transition probabilities

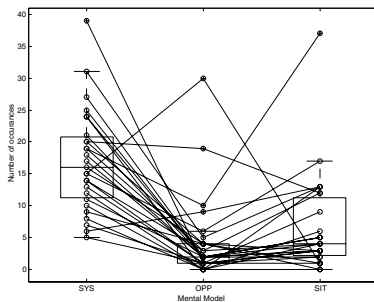


Fig. 5. Participants distribution to SYS, OPP and SIT events, summary for all three tasks

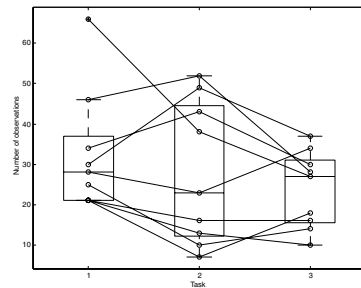


Fig. 6. Number of events per task and participant

3.3 Data Validation

In Table 4, information regarding the validity of the experiment instruments is presented. In summary, the size of the software is smaller (e.g., it was noted that there was almost no error handling) than in real life, but the way estimation takes place is similar to industrial practise, though one would only have performed estimation on systems larger than those in this study. The type of change tasks are sound given the functionality already present in the code provided. The think-aloud and interview setting are believed not to have affected the estimation process; in fact, one participant indicated that “it is common to have [a customer] breathing down one’s neck [while working]”. All participants believed that they understood the change tasks.

Table 4. Answers to questions assessing the validity of information given. N=No, Y=Yes, P=Partly

Question	P1	P3	P4	P5	P6	P7	P9	P10	P11
Do you consider the software	Y	P	P	P	N	Y	N	Y	Y

presented to be representative
for your work?

Do you consider the change Y Y Y Y Y Y Y Y Y
tasks to be representative?

Where you affected by the N N N P P N N N N
setting (think aloud, tape-
recorder)

During the experiment, the participants' estimates of effort required for each change task were collected. These estimates vary a lot. When asked about which activities were included in the estimates, it became clear that the participants differed in which activities were included in the estimates. For example, the individual inspection required by the think aloud instruction was not always included in the estimates given. Therefore, the suggested estimates are not analysed in this study, rather the focus is on the estimation processes.

The AFECS analysis has presented a number of challenges:

- Participants P3 and P4 spoke little while estimating. As seen in Figure 6, the number of encoded events differ a lot between the participants. Attempting to increase the number of phrases, for example by more training in how to think aloud, could possibly have changed the estimation process from current industrial practise. Therefore, no attempts to draw conclusions from the total number of events of a participant have been done.
- The fact that there are differences in the number and order of utterances can not necessarily be taken as a sign of that there is a difference in the estimation processes, except at a very detailed level. It is also so, that there may be events that are not captured through protocol analysis. For example, it can be assumed that all participants used the identifier names as a source of information for understanding what a section in code does. As seen in Table 3, this is not detected in all transcripts. Therefore, there may be process elements that are not visible in protocol analysis as performed in this study.
- Sometimes it can be hard to transcribe an event, due to missing information in the verbalization of thoughts. Especially, AFECS has an elaborate scheme for coding different types of hypotheses. In this study, almost all hypotheses detected have been classified as hypothesis related to the solution of the problem.
- Some phrases uttered during the think-aloud estimation are incomplete. For example, "Now I'll look at ..." followed by silence. This particular phrase has been encoded as the reading of code (SYS.ACT.INF.COD.REA), but not using finer AFECS coding than that.
- Sometimes a hypothesis is abandoned without this being explicitly stated. We have attempted to insert an abandon hypothesis AFECS code where it is believed that the hypothesis was abandoned. However, there is a large insecurity in this. This introduces some inaccuracy in the transfer probabilities in Figure 4. This problem has also been identified in [17], and it is also contrasted to [4], who found that programmers rarely abandon a hypothesis. The phenomenon that hypotheses are abandoned is also identified in [25]. Because the participants in this study were not trained in verbalizing why a hypothesis was abandoned, we cannot with confidence explain why hypotheses seem to be abandoned without any utterance stating so.

AFECS has not only provided challenges. The extensibility possibility has been used to augment AFECS with a new event that describes “estimate given”: SYS.ACT.SOL.

4 Analysis and Application of Results

From reading the protocols from the estimations, visualizing the AFECS transcriptions, and looking at Table 3, the estimation processes performed can be approximated by the following.

All estimations begun by the participant reading what new functionality was needed. The functional requirements were considered complete and consistent, but quality requirements such as required “code quality” and performance must have been assumed, since those were not present. One of the participants explicitly noted this.

Next, all participants built a primary top-level hypothesis regarding what kind of changes were needed, and then went on to see where those changes should be located in code. Several tactics for building the primary hypothesis have been visible:

- Three participants used an MSC to construct this hypothesis and get an initial understanding of the present control flow.
- Three participants built their primary hypothesis from experience without looking at code; for example, they assumed two changes: a) a user interface code change; and b) a change to logics-code.
- The remaining participants browsed the code to determine the control flow and thus build their primary hypothesis.

Two of the four participants that were using the more complex design B expressed that it was hard to comprehend the rationale for the design. All but one of the participants in Table 3 gave verbal evidence of that English-language identifier names were used to understand functionality, as well as to name a location in code. This implies that change effort estimation possibly can be facilitated already during design and coding by describing the design rationale and by using explanatory identifier names. The need to identify a location in code has also been noted in e.g. [18].

After the primary hypotheses were built, all participants used the code to verify and if necessary change their hypotheses. In Table 3, it can be seen that simulation of program execution has been clearly observed in 21 out of 24 estimations. The scenario simulated was the scenario described in the change task. The verification of the primary hypothesis lead to the identification of each location where a change in code would be necessary. Five of the participants wrote lists to keep track of needed changes.

After walking through and refining the initial effort estimate, three participants reviewed their estimates an extra time to further verify the estimates.

Five of the participants verbalized that they used the number of changes as an input to their estimation. Two participants also expressed that they took the perceived complexity of changes into account. From interviews after the estimations, it is clear that not all participants translated the required code changes into a set of actions required to perform the change (e.g. coding, inspecting, testing). This resulted in very different

estimates for each task, suggesting that an estimation practise should make clear what actions are required in an effort estimate.

A number of observations have been made during the AFECS analysis: All participants shift frequently among the three mental models that AFECS recognizes. In the example transcription in Figure 3, it can be seen that it is common to have more than one encoded event in a row that belong to the same mental model. This is also visible in Figure 4, where the average transfer probabilities for going from one mental model to another is the highest for going to the same mental model. This is interpreted as that the participants try to pursue one line of thought until they are hindered by the lack of other kind of information. Tools should take this into account, and support easy retrieval of knowledge, and shifts between various sources of information, and level of detail in those. Similar findings have been presented in [12].

In Figure 6, it can be seen that the total number of events differ a lot depending on the participant. This suggests that the amount of verbalized thoughts differ a lot between the participants, or that the individual estimation process differ a lot. From listening to the recordings, it is our belief that both can be true. Two participants spoke little despite being asked to “keep talking”, while other participants spoke almost without any pause. This suggests that protocol analysis can yield results with low validity if experiment participants for any reason do not naturally keep verbalizing their thoughts.

According to [11], one of the properties of the AFECS coding scheme is that it should be possible to extend on the fly, without having to re-transcribe each protocol at every change. During the course of analysis, a code for “estimate presented” has been added. This has been done without having to re-transcribe any protocol.

5 Conclusions and Future Work

Eleven industry professionals from two organizations producing small Internet-related software have participated in interviews regarding their project effort estimation practises. Nine of the participants have verbalized their thoughts while estimating the effort required to make three consecutive well-defined changes to a small Java program provided. The verbalized estimation processes have been analysed through protocol analysis using the AFECS coding scheme [11], which codes the Integrated Comprehension Model [15]. Higher level patterns have been manually searched for.

Though there are similarities in the verbalized estimation processes, they are also sufficiently different to motivate further investigation. For example, this study is not extensive enough to allow the detection of which thought patterns that are more efficient, or yield better estimates.

Several participants marked directly in code where changes were likely to be. Thus tools supporting marking code with temporary quality information such as “likely to change due to change request #15” may be beneficial. Since all participants assessed their hypotheses regarding change by mentally simulating the execution of the code, the ability to trace and possibly back-track the execution of a given scenario in code may be beneficial.

The AFECS coding scheme has been used to code behaviour during when comprehending a program for the purpose of estimating effort, without having to change the scheme, more than a minor addition. However, protocol analysis of verbalized thoughts is vulnerable in terms of that not all thoughts are necessarily verbalized. It is concluded that AFECS can be used for understanding task level effort estimation behaviour, but one should be aware of that not all processes elements necessary are captured. Therefore, this study can only be regarded as an initial study of individual task level effort estimation strategy. Training participants further in how to verbalize their thoughts may reveal more process elements, but at the risk of affecting the participants in how they work. A possible way of balancing the risk of not capturing all process elements is to ask the participant to describe their solution strategy after performing an estimation, and combining this information with the protocol.

In this study, it was seen that two of the participants explicitly compared their estimates to those earlier made. This suggests that estimation by analogy may be worth investigating further. The Analytical Hierarchical Process (AHP) [21] has been used to successfully to estimate the size of implementations for some small algorithms [19], by pair-wise comparing an estimate to multiple other known sizes. Attempting to use the AHP also for effort estimation is subject to further research.

References

1. Arisholm, E., Sjøberg, D.I.K., Jørgensen, M. "Assessing the Changeability of two Object Oriented Design Alternatives -A Controlled Experiment". Empirical Software Engineering, accepted for publication. 2001
2. Bratthall, L., Runeson, P., Adelswärd, K., Eriksson, W. "A Survey of Lead-time Challenges in the Development and Evolution of Distributed Real-time Systems". Information and Software Technology, Vol. 42, No. 13, pp. 947-958. September, 2000
3. Briand, L.C., El Emann, K., et al. An Assessment and Comparison of Common Software Cost Estimation Techniques. ISERN. 1998
4. Brooks, R. "Towards a Theory of the Comprehension of Computer Programs". Int.l J. of Man-Machine Studies. Vol. 18, pp. 543-554. 1983
5. Brown, N.R., Siegler, R.S. "The Role of Availability in the Estimation of National Populations." Memory and Cognition, Vol. 20, pp. 406-412. 1993
6. Cook, J.E., Wolf, A.L. "Automating Process Discovery through Event-Data Analysis" Proc. 17th International Conference on Software Engineering. Seattle, Washington, USA. April, 1995
7. Delaney, W.A. "Predicting the Costs of Computer Programs". Data Processing Magazine, pp. 32-34. 1966
8. Eriksson, K.A., Simon, H.A. Protocol Analysis: Verbal Reports as Data. 3rd printing. 1999
9. Jørgensen, M., Sjøberg, D., Kirkebøen, G. "The Prediction Ability of Experienced Software Maintainers". In Proc. 4th European Conference on Software Maintenance and Reengineering. Zürich, Switzerland. 2000
10. Letovsky, S. "Delocalized Plans and Program Comprehension". IEEE Software, pp. 41-49. May, 1986
11. von Mayrhauser, A., Lang, S. "A Coding Scheme to Support Systematic Analysis of Software Comprehension". IEEE Trans. on Software Engineering. Vol. 25, No. 4, pp. 526-540. July/August, 1999

12. von Mayrhauser, A., Vans, A.M. "From Program Comprehension to Tool Requirements for an Industrial Environment". In Proc. Second Workshop on Program Comprehension, pp. 78-86. IEEE Computer Society. July, 1993
13. von Mayrhauser, A., Vans, A.M. "Dynamic Code Cognition Behaviours for Large Scale Code". In Proc. Third Workshop on Program Comprehension, pp. 74-81. IEEE Computer Society. November, 1994
14. von Mayrhauser, A., Vans, A.M. "Comprehension Processes During Large Scale Maintenance". In Proc. 16th Intl. Conf. Software Engineering. Sorrento, Italy. May, 1994
15. von Mayrhauser, A., Vans, A.M. "Industrial Experience with an Integrated Code Comprehension Model". Software Engineering Journal, pp. 171-182. September, 1995
16. von Mayrhauser, A., Vans, A.M. "Identification of Dynamic Comprehension Processes During Large Scale Maintenance". IEEE Transactions on Software Engineering, Vol. 22, No. 6, pp. 424-437. 1996
17. von Mayrhauser, A., Vans, A.M. "On the Role of Hypotheses During Opportunistic Understanding While Porting Large Scale Code". In Proc. Fourth Workshop on Program Comprehension, pp. 68-77. IEEE Computer Society. March, 1996
18. von Mayrhauser, A., Vans, A.M., Howe, A.E. "Program Understanding Behaviour during Enhancement of Large-scale Software". Journal of Software Maintenance, Vol. 9, pp. 299-327. 1997
19. Miranda, E. "An Evaluation of the Paired Comparisons Method for Software Sizing". Int'l Conf. Software Engineering. pp. 597-604. Limerick, Ireland. 2000
20. Myrtveit, I., Stensrud, E. "A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models". IEEE Transactions on Software Engineering, Vol. 25, pp. 510-525. 1999
21. Saaty, T.L. The Analytic Hierarchy Process. McGraw-Hill, New York, USA. 1980
22. Soloway, E., Adelson, B., et al. (Eds.) Knowledge and Process in the Comprehension of Computer Programs. The Nature of Expertise. Lawrence Erlbaum Assoc. 1988
23. Standish Group. <http://standishgroup.com/visitor/chaos.htm>. Accessed 20 Feb. 2001
24. Storey, M.D., Wong, K., Müller, H.A. "How do Program Understanding Tools Affect How Programmers Understand Programs?". Proc. Fourth Working Conf. Reverse Engineering. Amsterdam, The Netherlands. October, 1997
25. Vans, A.M., von Mayrhauser, A., Somlo, G. "Program Understanding Behaviour during Corrective Maintenance of Large-scale Software". International Journal of Human-Computer Studies, Vol. 51, pp. 31-70. 1999
26. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers. 1999

Evaluation of a Business Application Framework Using Complexity and Functionality Metrics

Hikaru Fujiwara¹, Shinji Kusumoto¹, Katsuro Inoue¹, Toshifusa Ootsubo², and
Katsuhiko Yuura²

¹Graduate School of Engineering Science, Osaka University 1-3 Machikaneyama,
Toyonaka, Osaka, 560-8531, Japan Tel: +81-6-6850-6571, Fax: +81-6-6850-6574

²Business Solution Systems Division, Hitachi Ltd.
h-fujiwr@ics.es.osaka-u.ac.jp

Abstract. This paper experimentally evaluates the usefulness of a business application framework from a viewpoint of saving cost and quality of the software in a company. Here, we conducted two case studies. In the case studies, four kinds of applications are developed. Each of them is developed in two ways: using framework-based reuse and conventional module-based reuse. Then, we evaluate the difference among them using the several functionality and complexity metrics. As the results, the framework-based reuse would be more efficient than the module-based reuse in the company.

1. Introduction

As the size and the complexity of software increase, it becomes important to develop high-quality software cost-effectively within a specified period. In order to attain it, several software engineering techniques have been developed. Reuse is one of the most famous techniques among them.

The definition of reuse is the use of an existing software component in a new context, either elsewhere in the same system or in another system [4]. It is generally said that software reuse is improved productivity and quality, resulting in cost saving. Several research studies have demonstrated the actual benefits[2][7][9].

On the other hand, object-oriented development intends to construct the software by reusing several software components which have high cohesion and are easy to combine. By reusing the components which have high quality, the improvement of the quality of the software will be attained and development period will be also decreased [8]. As in development with object-oriented languages, developers reuse a particular library called a framework [3]. A framework is collection of classes that provide a set of services for a particular domain; a framework thus exports a number of individual classes and mechanisms which clients can use or adapt. In a typical development with the framework, at first, developers take out the primitive parts of a program. Then, they develop the other parts needed for the developing application, and combine them into a program. The Microsoft Foundation Classes (called MFC) is a framework for building applications that conform to the Microsoft Windows user interface standards.

In a department of Hitachi Ltd., a business application framework for a specific domain has been developed and introduced. However, it is difficult to transfer the new framework to the development because developers in the department use the original reuse technique that is a conventional module-based reuse. From a viewpoint of saving cost and improving the productivity, it is clear that framework-based reuse is more appropriate than the module-based reuse. For effective technical transfer, it is necessary to motivate the developers who will use the new technique by showing the benefit of using it quantitatively.

This paper experimentally evaluates the usefulness of the framework from a viewpoint of saving cost and quality of the software. Here, we conducted two case studies. In the case studies, four kinds of applications are developed. Each of them is developed in two ways: using framework-based reuse and conventional module-based reuse. Then, we evaluate the difference among them using the several functionality and complexity metrics. As the results, the framework-based reuse would be more efficient than the module-based reuse in the department.

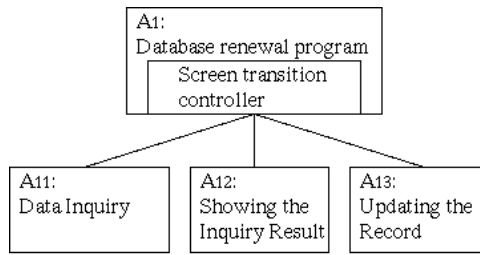
Section 2 introduces the characteristics of the target application software and the proposed framework. Then, Section 3 describes the case studies and Section 4 analyzes the empirical results. Finally, Section 5 concludes and mentions about future research topics.

2. Preliminary

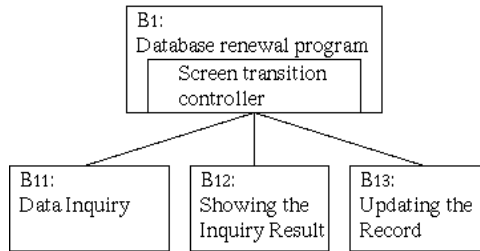
2.1 Target Application Software

In a department of Hitachi Ltd., a series of on-line application software for a local government has been developed. Here, “series” implies that there exist multiple projects where similar applications are successively developed for several local governments. The functions of the application software includes such as a family registration, payment of various taxes, treatment of health insurance. Since the data handled by each local government are different, the details of each applications are different, but the fundamental processing of each function is the same. To put it simply, it controls the processing of conventional database operations.

So far, to develop the series of the applications, conventional module-based reuse has been used. Figure 1 shows the typical example. Figure 1 (a) is the module structure of an application that makes a new record for a health insurance for a local government A. Module A_1 is a main module and controls the transition of the screens. Users manipulate the application through the screens that are changed based on the users' input. Each of the modules A_{11} , A_{12} and A_{13} corresponds to the process of data inquiry, showing the inquiry result and updating the record. In the conventional module-based reuse, each of the modules is reused in the next development. Figure 1 (b) shows the module structure of the same application for a local government B. Here, modules B_{11} , B_{12} and B_{13} are reused by modifying the corresponding modules A_{11} , A_{12} and A_{13} . The reused modules are modified in a skillful way in the department.



(a) Application for a local government A



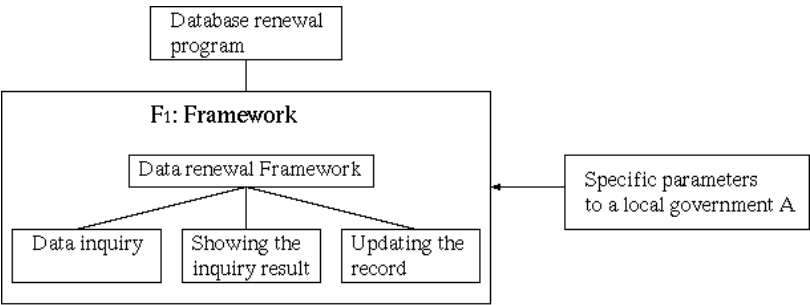
(b) Application for a local government B

Fig. 1. Conventional module-based reuse

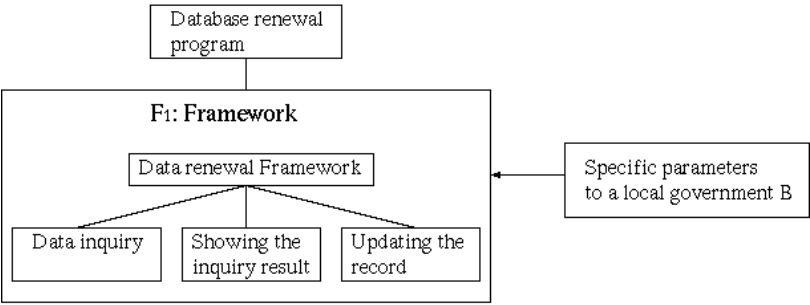
2.2 Proposed Framework

Recently, many companies have started to introduce object-oriented technology into their software development environments. The application software described in Section 2.1 will be also developed by object-oriented approach and implemented using Java. So, the new framework is going to be introduced. The framework is intended to reuse the processing of the transition of the screens in addition to the module-based reuse. In the framework, typical transitions of screens are prepared, that are data inquiry, data renewal, data addition, data removal and so on. For example, data renewal consists of the three kinds of screen manipulations: data inquiry, showing the inquiry result and updating the record.

For example, Figure 2 explains the framework-based reuse. Figure 2 (a) shows the application software that makes a new record for a health insurance for a local government A. F_1 is a framework where a screen transition pattern for data renewal is got together. By using F_1 , the processing of data inquiry, showing the inquiry result and updating the record are simply implemented together. The specific information to the local government A is given by parameters. So, in case of developing the similar application for a local government B, it is easily done using the framework F_1 and the specific information to the local government B shown in Figure 2(b).



(a) Application for a local government A



(b) Application for a local government B

Fig. 2. Framework-based reuse

3. Case Studies

Here, we explain the case studies to evaluate the usefulness of the proposed framework.

3.1 Design of Case Study

We conducted to case studies: Case studies 1 and 2. In case study 1, four kinds of programs are developed in two ways: conventional and framework-based reuse. Here, C_a , C_b , C_c and C_d represent the four kinds of programs developed using the framework-based reuse. On the other hand, P_a , P_b , P_c and P_d represent the ones developed using the conventional reuse. The functions implemented in C_i and P_i ($i=a,b,c,d$) are the same. We compare C_i and P_i from the viewpoint of the productivity and quality.

In case study 2, a program is also developed in two ways. In this case, each of four functions (called f_a , f_b , f_c and f_d) are continuously added to the program. That is, using the framework reuse, at first, a program (called C_a), includes only f_a , is developed and then a program (called C_{a+b}) is developed by adding the function f_b to C_a . Similarly,

C_{a+b+c} is developed by adding f_c to C_{a+b} and finally $C_{a+b+c+d}$ is completed by adding f_d to C_{a+b+c} . On the other hand, using the conventional reuse, programs P_a , P_{a+b} , P_{a+b+c} and $P_{a+b+c+d}$ are developed. The functions between C_i and P_i ($i=a, a+b, a+b+c, a+b+c+d$) are the same. We compare the differences between the two successive programs from the viewpoint of the productivity and quality.

3.2 Metrics

Unfortunately, we could not collect the actual development effort and the number of faults injected in the development. So, we used the following metrics to indirectly evaluate the productivity and quality: OOFP (Object-Oriented Function Point)[5] and C&K metrics (Chidamber and Kemerer's metrics)[6].

Function point (FP) was proposed to help measure the functionality of software systems and to estimate the effort required for the software development. However, the original function point is not proposed to object-oriented program. Recently, OOFP has been proposed in [5] and relatively easy to calculate from the programs. So, we used the OOFP to indirectly measure the productivity.

On the other hand, C&K metrics is one of the most famous metrics to evaluate the complexity of object-oriented software. Several studies reported the relationship between the values of C&K metrics and the number of faults injected. For example, Basili et al. empirically evaluated that C&K metrics show to be better predictors of fault-proneness of the class than the traditional code metrics [1]. Thus, we used C&K metrics to indirectly measure the quality of the applications.

3.3 OOFP

Caldiera et al. have defined an adaptation of traditional FP, called OOFP, to enable the measurement of object oriented analysis and design specifications[5].

In traditional developments, the central concepts used in counting function points are *logical files* and *transactions* that operate on those files. In OO systems, the core concept is no longer related to file or data bases; instead the central concept is the "object". A class is the candidate for mapping logical files into the OO paradigm. In the FP method, logical files (LFs) are divided into *internal* logical files (ILFs) and *external* interface files (EIFs). Classes within the application boundary correspond to ILFs. Classes outside the application boundary correspond to EIFs. For each ILF/EIF, it is necessary to compute the number of DETs (Data Element Types) and RETs (Record Element Types). Simple attributes, such as integers and strings, are considered as DETs. A complex attribute, such as an attribute whose type is a class or a reference to another class, is considered as RET.

Transactions in FP method are classified as *inputs*, *outputs* and *inquiries*. In OOFP, they are simply treated as generic Service Requests (SRs). Each service request (method) in each class in the system is examined. Abstract methods are not counted. Concrete methods are only counted once (in the class they are declared), even if they are inherited by several subclasses, because they are only coded once. If a method is to be counted, the data types referenced in it are classified: (1) simple

items are simple data items (such as integers and strings) referenced as arguments of the method, and simple global variables referenced in the method (2) complex items are complex (class or reference to class) arguments, objects or complex global variables referenced by the method. Simple items are considered as DETs, and Complex items are considered as File Types Referenced (FTRs).

Finally, OOFP is calculated by summing up the number of ILF, EIF and SR weighted by each complexity based on DET, RET and FTR.

3.4 C&K Metrics

Chidamber and Kemerer's metrics are based on measurement theory, and reflect the viewpoints of experienced object-oriented software developers. Chidamber and Kemerer defined the following six metrics[6]:

1. WMC(Weighted Method per Class) : Assume that a class C includes methods M_1, \dots, M_n . Let $c_i (i = 1..n)$ be the static complexity of method M_i . Then, $WMC(C) = \sum_{i=1}^n c_i$. If it can be supposed that all methods of the given class C are equally complex, then $WMC(C)$ is simply the number of methods defined in the class C [1].
2. DIT(Depth of Inheritance Tree of a class) : $DIT(C)$ is the depth of class C in the inheritance tree of a given program. If it can be supposed that the whole inheritance graph is a tree, then $DIT(C)$ is the length of path from the root to the class C [1].
3. NOC(Number Of Children) : $NOC(C)$ is the number of immediate sub-classes subordinated to a class C in the class hierarchy of a given program.
4. CBO(Coupling Between Object class) : $CBO(C)$ is the number of couplings between a class C and any other class. The coupling means that the class C uses member functions and/or instance variables in any other class.
5. RFC(Response For a Class) : Let $M_s(C)$ be a set of methods in a class C , and define $M_r(C) = \{M_j \mid M_j \text{ is a method called by any } M_i \notin M_s(C) \text{ and } M_j \notin M_s(C)\}$. Then $RFC(C) = |M_r(C) \cup M_s(C)|$.
6. LCOM(Lack of Cohesion in Methods) : Assume that a class C includes methods M_1, M_2, \dots, M_n . For each $i (i = 1..n)$, let I_i be a set of instance variables used by method M_i . $LCOM(C)$ is defined as the number of such pairs of method (M_i, M_j) that $I_i \cap I_j = \emptyset$, minus such pairs of method (M_k, M_l) that $I_k \cap I_l \neq \emptyset$, when the former is greater than the latter. Otherwise, $LCOM(C)$ is defined to be zero.

3.5 Application of the Metrics

Here, we explain how to apply the metrics to the programs. In order to evaluate the effect of the framework, we pay attention to the newly developed part of the programs. Figure 3 shows rough structure of the C_a and P_a in case study 1. In C_a , FW represents the framework. On the other hand, Figure 4 shows rough structure of the C_a , P_a , C_{a+b} and P_{a+b} in case study 2. In both cases, the values of metrics are calculated from the newly developed part of the programs (shaded portion).

Tables 1 and 2 show the measurement results of case studies 1 and 2. The values of C&K metrics are the average values per a class. Here, the values of NOC and DIT are omitted since these applications were developed without class inheritance and the values became 0.

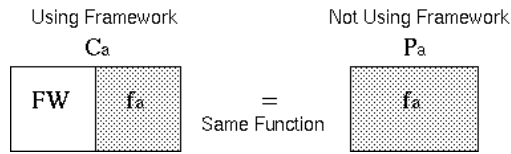


Fig. 3. Measurement in case study 1

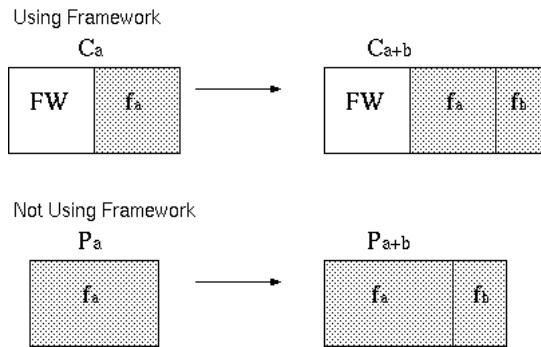


Fig. 4. Measurement in case study 2

4. Analysis

4.1 Case Study 1

With respect to OOFp, the values of programs developed using framework are smaller than ones of programs developed using conventional reuse¹. Especially, the values of C_a , C_b and C_d are much smaller than ones of corresponding programs.

Next, we analyze the values of C&K metrics:

- CBO, RFC: The values of C_i are higher complexity than ones of P_i . It means that the newly developed classes of C_i include a lot of method calls. We examined the classes of C_i that have high CBO and RFC values and found that the called method are mostly included in the classes in the framework. So, if the classes of the framework have high quality, the complexity does not affect the quality of the overall application programs.

¹ Since P_i s have been developed scratch, they actually had no reused part.

- WMC, LCOM: The values of C_i are higher than ones of P_i . We also examined the classes of C_i that have high WMC and LCOM values and found that there are many simple methods, that set/get the values of attributes in the class of framework. The size of these methods is about one or two lines of codes. Therefore, the complexity does not affect the quality of the overall application programs, too.

Table 1. Result in case study 1

	OOFP	CBO	RFC	WMC	LCOM
C_a	176	3.8	14.4	7.4	21.4
C_b	180	5.8	18.2	7.6	23.2
C_c	418	5.4	33.1	8.1	28.7
C_d	252	4.1	17.8	6.4	13.8
P_a	526	2.1	5.8	3.3	2.1
P_b	526	2.3	5.9	3.3	2.1
P_c	671	3.0	7.4	3.4	2.0
P_d	672	2.4	8.6	4.3	15.7

4.2 Case Study 2

Using Table 2, we evaluate the amount of increased values of the metrics. Here, Δ_b , Δ_c , Δ_d represent the increased values of OOFP by adding the function f_b , f_c and f_d , respectively. For C_i (using the framework), the values of Δ_b , Δ_c and Δ_d are 75, 327 and 165, respectively. On the other hand, for P_i (not using the framework), the values of Δ_b , Δ_c and Δ_d are 89, 234 and 235. For Δ_c , the values of C_i has higher value than ones of P_i . This is because in order to implement the function f_c , a lot of data items must be handled compared to the function f_b and f_d . Also, it is found that the adjustment between the function f_c and the framework is not good. One way to cope with it is to add the new components to the framework to fit the function like fc .

Next, we analyze the values of C&K metrics. Here, $(\delta_b, \delta_c, \delta_d)$ represent the increased values of C&K metrics by adding the function f_b , f_c and f_d , respectively.

- CBO, WMC: There is few difference between the increased values in C_i and P_i .
- RFC: The increased value of δ_c in C_i is 13.4 (= 30.1 - 16.7) and much larger than one (1.7 = 8.6 - 6.9) in P_i . Using the framework, the number of method call is increased. The called method are mostly included in the classes in the framework. So, if the classes of the framework have high quality, the complexity does not affect the quality of the overall application programs.
- LCOM: There are not so much difference among C_i . On the other hand, the increased value in δ_d (8.2 = 10.0 - 1.8) in P_i is larger than in δ_b and δ_c .

For $C_{a+b+c+d}$ and $P_{a+b+c+d}$, the values of OOFP are 743 and 1084. So, totally, by using the framework, the effort of the development would be reduced. On the other hand, with respect to C&K metrics, the complexity of $C_{a+b+c+d}$ is higher than $P_{a+b+c+d}$. However, the complexity is caused by the interaction among the newly developed class and the

framework classes. So, we consider that if the framework has high quality, the complexity does not affect the quality of the overall application programs.

Table 2. Result in case study 2

	OOFP	CBO	RFC	WMC	LCOM
C_a	176	3.8	14.4	7.4	21.4
C_{ash}	251	5.0	16.7	7.6	20.1
C_{ashac}	578	5.9	30.1	8.3	28.6
$C_{ashacsd}$	743	5.8	28.3	7.9	25.6
P_a	526	2.1	5.8	3.3	2.1
P_{ash}	615	2.8	6.9	3.3	2.0
P_{ashac}	849	3.7	8.6	3.3	1.8
$P_{ashacsd}$	1084	4.0	10.5	3.9	10.0

5. Conclusion

We have experimentally evaluated the usefulness of the framework from a view point of the effectiveness of the saving cost and the quality of the software. As the results of two case studies, the framework-based reuse is more efficient than the module-based reuse in the department.

Actually, the value of FW's OOFP is 1298. However, as the result of case study 1, the value of OOFP in framework-based reuse about 2.5 times as effective as in conventional reuse. So, it is expected that the department developing the series of project will save the effort after three or four applications have developed, whereas the investment for FW was spent.

In order to show the usefulness of the framework, we are going to apply the framework to many software development projects. In addition, it is necessary to add the new components to the framework to increase the applicability of it.

References

1. V. R. Basili, L. C. Briand and W. L. Melo: "A validation of object-oriented design metrics as quality indicators", IEEE Trans. on Software Eng. Vol. 20, No. 22, pp. 751-761 (1996).
2. V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: "The software engineering laboratory - an operational software experience", Proc. of ICSE14, pp. 370-381 (1992).
3. G. Booch: *Object-Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing (1994).
4. C. Braun: *Reuse*, in John J. Marciniak, editor, Encyclopedia of Software Engineering, vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).
5. G. Caldiera, G. Antoniol, R. Fiutem and C. Lokan: "Definition and experimental evaluation of function points for object-oriented systems", IEEE, Proc. of METRICS98, pp.167-178 (1998).
6. S. R. Chidamber and C. F. Kemerer: "A metrics suite for object-oriented design", IEEE Trans. on Software Eng., Vol. 20, No. 6, pp. 476-493 (1994).

7. S. Isoda: "Experience report on a software reuse project: Its structure, activities, and statistical results", Proc. of ICSE14, pp.320-326 (1992).
8. I. Jacobson, M. Griss and P. Jacobson: *Software Reuse ---Architecture Process and Organization for Business Success---*, Addison-Wesley (1997).
9. B. Keepence and M. Mannion: "*Using patterns to model variability in product families*", IEEE Software, Vol. 16, No. 4, pp. 102-108 (1999).

Author Index

Allenby, Karen.....	210	Lebsanft, Karl.....	78
Anderson, Stuart.....	27	Lepasaar, Marion.....	68
Araujo, Renata.....	297	Lichter, Horst.....	227
Arisholm, Erik.....	356	Lindvall, Mikael.....	110
Bardill, Mike.....	210	Mäntyniemi, A.....	57
Basili, Victor.....	3, 110	Marschall, Frank.....	326
Becker-Kornstaedt, Ulrike.....	312	Maurer, Frank.....	255
Borges, Marcos.....	297	McDermid, John.....	210
Bratthall, Lars.....	356	Mellis, Werner.....	4
Burton, Simon.....	210	Moe, Nils Brede.....	167
Buttle, Darren.....	210	Münch, Jürgen.....	255
Costa, Patricia.....	110	Murdoch, John.....	210
Dekkers, Ton.....	240	Nättinen, Minna.....	141
Dingsøyr, Torgeir.....	167	Nagel, Christof.....	153
Felici, Massimo.....	27	Nedstam, Josef.....	42
Fröhlich, Peter.....	227	Nilsson, Jennie.....	42
Fujiwara, Hikaru.....	371	Nytrø, Øystein.....	167
Gnatz, Michael.....	326	Ohlsson, Magnus C.....	341
Henninger, Scott.....	182	Ootsubo, Toshifusa.....	371
Höst, Martin.....	42	Parviainen, Päivi.....	141
Hutchesson, Stuart.....	210	Piattini, Mario.....	86
Iivari, Netta.....	98	Pikkarainen, Minna.....	57
Inoue, Katsuro.....	371	Polo, Macario.....	86
Jaakkola, Hannu.....	68	Popp, Gerhard.....	326
Jaccheri, Letizia M.....	271	Rahikkala, Tua.....	57
Jeffery, Ross.....	6	Rausch, Andreas.....	326
Jiménez, Mar.....	86	Regnell, Björn.....	42
Jokela, Timo.....	98	Rombach, Dieter.....	1
Jørgensen, Magne.....	356	Ruiz, Francisco.....	86
Kauppinen, Marjo.....	196	Runeson, Per.....	341
Komi-Serviö, Seija.....	57	Rus, Ioana.....	110
Kucza, Timo.....	141	Schlabach, Jason.....	182
Kujala, Sari.....	196	Schneider, Kurt.....	126
Kusumoto, Shinji.....	371	Schwerin, Wolfgang.....	326
		Seppänen, Veikko.....	57
		Shaw, Mary.....	5
		Shull, Forrest.....	110

Stålhane, Tor..... 271
 Stephenson, Alan.....210
 Störrle, Harald..... 282

 Tesoriero, Roseanne..... 110

 Varkoi, Timo..... 68

Wohlin, Claes..... 341
 Wong, Bernard..... 6
 Wong, Les..... 255

 Yuura, Katsuhiko.....371

 Zelkowitz, Marvin..... 110
 Zeller, Manfred.....227
 Zettel, Jörg.....255